CHAPTER 1

INTRODUCTION

1.1 MOTIVATION

Developing a defect free software system is very difficult and most of the time there are some unknown bugs or unforeseen deficiencies even in software projects where the principles of the software development methodologies were applied carefully. Due to some defective software modules, the maintenance phase of software projects could become really painful for the users and costly for the enterprises. That is why, predicting the defective modules or files in a software system prior to project deployment is a very crucial activity, since it leads to a decrease in the total cost of the project and an increase in overall project success rate.

Defect prediction will give one more chance to the development team to retest the modules or files for which the defectiveness probability is high. By spending more time on the defective modules and no time on the non-defective ones, the resources of the project would be utilized better and as a result, the maintenance phase of the project will be easier for both the customers and the project owners.

1.2 PROBLEM DEFINITION

In defect prediction literature, there are many defect prediction algorithms studied like regression, Ekanayake, rule induction, decision tree approaches like C4.5, case-based reasoning (CBR), artificial neural networks, linear discriminant analysis, *k*-nearest neighbour, *k*-star, Bayesian networks and support vector machine based classifiers. According to the no free lunch theorem, there is no algorithm which is better than other algorithms on all data sets. That is why, most of the time it is difficult to generalize the performance of one algorithm and say that it is the best technique for defect prediction. We need to develop more reliable research procedures before we can have confidence in the conclusion of comparative studies of software prediction models. The accuracy of a specific defect prediction method is very much dependent on the attributes of the data set like its size, number of attributes and distribution. That is why, it is better to ask which method is the best in a specified context rather than asking which one is the best in general.

1.3 OBJECTIVE OF PROJECT

- This project reviews the use of Bayesian networks (BNs) in predicting software defects which is identifying error prone software modules referred to as software defect prediction.
- In this project, we build a Bayesian network among metrics and defectiveness, to measure which metrics are more important in terms of their effect on defectiveness and to explore the influential relationships among them. As a result of learning such a network, we find the defectiveness probability of the whole software system, the order of metrics in terms of their contribution to accurate prediction of defectiveness, and the probabilistic influential relationships among metrics and defectiveness.

1.4 PROBLEMS IN EXISTING SYSTEM

In earlier methods, static code features were used more. But afterwards, it was understood that beside the effect of static code metrics on defect prediction, other measures like process metrics are also effective and should be investigated. Static code measures alone are not able to predict software defects accurately.

In defect prediction literature, there are many defect prediction algorithms studied like regression, Ekanayake, rule induction, decision tree approaches like C4.5, case-based reasoning (CBR), artificial neural networks, linear discriminant analysis, *k*-nearest neighbour, *k*-star, Bayesian networks and support vector machine based classifiers. According to the no free lunch theorem, there is no algorithm which is better than other algorithms on all data sets. That is why, most of the time it is difficult to generalize the performance of one algorithm and say that it is the best technique for defect prediction. We need to develop more reliable research procedures before we can have confidence in the conclusion of comparative studies of software prediction models. The accuracy of a specific defect prediction method is very much dependent on the attributes of the data set like its size, number of attributes and distribution. That is why, it is better to ask which method is the best in a specified context rather than asking which one is the best in general.

Looking at the defect prediction problem from the perspective that all or an effective subset of software or process metrics must be considered together besides static code measures, Bayesian network model is a very good candidate for taking into consideration several process or product metrics at the same time and measuring their effect.

To remedy these problems, we build a Bayesian network among metrics and defectiveness, to measure which metrics are more important in terms of their effect on defectiveness and to explore the influential relationships among them. As a result of learning such a network, we find the defectiveness probability of the whole software system, the order of metrics in terms of their contribution to accurate prediction of defectiveness, and the probabilistic influential relationships among metrics and defectiveness.

1.5 ORGANIZATION OF DOCUMENTATION

In this project documentation we have initially put the definition and objective of the project as well as the design of the project which is followed by the implementation and testing phases. Finally the project has been concluded successfully and also the future enhancements of the project were given in this documentation.

CHAPTER 2

LITERATION SURVEY

2.1 BAYESIAN NETWORKS

A Bayesian network is a directed acyclic graph (DAG), composed of *E* edges and *V* vertices which represent joint probability distribution of a set of variables. In this notation, each vertex represents a variable and each edge represents the causal or associational influence of one variable to its successor in the network.

Let $X = \{ X_1 , X_2 , \dots X_n \}$ be n variables taking continuous or discrete values. The probability distribution of X_i is shown as $P(X_i | a_{xi})$ where a_{xi} 's represent parents of X_i if any. When there are no parents of X_i , then it is a prior probability distribution and can be shown as $P(X_i)$.

The joint probability distribution of X can be calculated using chain rule:

$$\begin{aligned} P(X) &= P(X_1 | X_2 , X_3 , \dots, X_n) P(X_2 , X_3 , \dots, X_n) \\ &= P(X_1 | X_2 , \dots, X_n) P(X_2 | X_3 , \dots, X_n) P(X_3 , \dots, X_n) \\ &= P(X_1 | X_2 , \dots, X_n) P(X_2 | X_3 , \dots, X_n)\dots P(X_{n-1} | X_n) P(X_n) \\ &= \mathop{}_{i=1}^{N} P(X_i | X_{i+1} , \dots, X_n) \end{aligned} \qquad (1)$$

Given the parents of X_i , other variables are independent from X_i , so we can write the joint probability distribution as

$$P(X) = \mathop{}_{i=1}^{N} P(X_i |a_{xi}) \qquad (2)$$

On the other hand, Bayes' rule is used to calculate the posterior probability of X_i in a Bayesian network based on the evidence information present. We can calculate probabilities either towards from causes to effects ($P(X_i | E)$) or from effects to causes ($P(E| X_i)$). Calculating probability of effects from causes is called causal inference whereas calculating probability of causes from effects is called diagnostic inference.

2.2BAYESIAN NETWORK MODELS

2.2.1 MARKOV BLANKET

In machine learning, the Markov blanket for a node A in a Bayesian network is the set of nodes ∂A composed of A's parents, its children, and its children's other parents. In a Markov network, the Markov blanket of a node is its set of neighboring nodes. A Markov blanket may also be denoted by $MB(A)$.
Every set of nodes in the network is conditionally independent of A when conditioned on the set ∂A, that is, when conditioned on the Markov blanket of the node A. The probability has the Markov property; formally, for distinct nodes A and B:

$$\Pr(A \mid \partial A, B) = \Pr(A \mid \partial A).$$

The Markov blanket of a node contains all the variables that shield the node from the rest of the network. This means that the Markov blanket of a node is the only knowledge needed to predict the behavior of that node.
In a Bayesian network, the values of the parents and children of a node evidently give information about that node; however, its children's parents also have to be included, because they can be used to explain away the node in question.

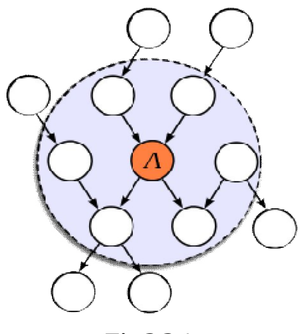


Fig 2.2.1

In a Bayesian network, the Markov blanket of node *A* includes its parents, children and the other parents of all of its children.

2.2.2 NAÏVE BAYES

Given the intractable sample complexity for learning Bayesian classifiers, we must look for ways to reduce this complexity. The Naive Bayes classifier does this by making a conditional independence assumption that dramatically reduces the number of parameters to be estimated when modeling P(XjY), from our original $2(2^n\ 1)$ to just 2n.

A naïve Bayes classifier is a simple probabilistic classifier based on applying Bayes' theorem with strong (naive) independence assumptions. Naive Bayes classifier assumes that the presence or absence of a particular feature is unrelated to the presence or absence of any other feature, given the class variable. It only requires a small amount of training data to estimate the parameters (means and variances of the variables) necessary for classification. Because independent variables are assumed, only the variances of the variables for each class need to be determined and not the entire covariance matrix.

Abstractly, the probability model for a classifier is a conditional model

$p(C|F_1,\dots,F_n)$

over a dependent class variable C with a small number of outcomes or *classes*, conditional on several feature variables F_1 through F_n. The problem is that if the number of features n is large or when a feature can take on a large number of values, then basing such a model on probability table is infeasible.

Using Bayes' theorem, this can be written

$$p(C|F_1,\dots,F_n) = \frac{p(C)\ p(F_1,\dots,F_n|C)}{p(F_1,\dots,F_n)}.$$

In plain English, using Bayesian Probability terminology, the above equation can be written as

$$\text{posterior} = \frac{\text{prior} \times \text{likelihood}}{\text{evidence}}.$$

In practice, there is interest only in the numerator of that fraction, because the denominator does not depend on C and the values of the features F_i are given, so that the denominator is effectively constant. The numerator is equivalent to the joint probability model

$$p(C, F_1, \dots, F_n)$$

which can be rewritten as follows, using the chain rule for repeated applications of the definition of conditional probability:

$$\begin{aligned}
&p(C, F_1, \dots, F_n)\\
&= p(C)\ p(F_1, \dots, F_n|C)\\
&= p(C)\ p(F_1|C)\ p(F_2, \dots, F_n|C, F_1)\\
&= p(C)\ p(F_1|C)\ p(F_2|C, F_1)\ p(F_3, \dots, F_n|C, F_1, F_2)\\
&= p(C)\ p(F_1|C)\ p(F_2|C, F_1)\ p(F_3|C, F_1, F_2)\ p(F_4, \dots, F_n|C, F_1, F_2, F_3)\\
&= p(C)\ p(F_1|C)\ p(F_2|C, F_1)\ \dots p(F_n|C, F_1, F_2, F_3, \dots, F_{n-1}).
\end{aligned}$$

Now the "naive" conditional independence assumptions come into play: assume that each feature F_i is conditionally independent of every other feature F_j for $j \neq i$ given the category C. This means that

$p(F_i|C, F_j) = p(F_i|C)$, $p(F_i|C, F_j, F_k) = p(F_i|C)$, $p(F_i|C, F_j, F_k, F_l) = p(F_i|C)$, and so on,

for $i \neq j, k, l$, and so the joint model can be expressed as

$$\begin{aligned}
&p(C|F_1, \dots, F_n)\\
&\propto p(C, F_1, \dots, F_n)\\
&\propto p(C)\ p(F_1|C)\ p(F_2|C)\ p(F_3|C)\ \cdots\\
&\propto p(C) \prod_{i=1}^{n} p(F_i|C).
\end{aligned}$$

This means that under the above independence assumptions, the conditional distribution over the class variable C is:

$$p(C|F_1, \dots, F_n) = \frac{1}{Z} p(C) \prod_{i=1}^{n} p(F_i|C)$$

where Z (the evidence) is a scaling factor dependent only on $F_1, \dots, F_n$, that is, a constant if the values of the feature variables are known.

Models of this form are much more manageable, because they factor into a so-called *class prior* $p(C)$ and independent probability distributions $p(F_i|C)$. If there are k classes and if a model for each $p(F_i|C = c)$ can be expressed in terms of r parameters, then the corresponding naive Bayes model has $(k - 1) + n\, r\, k$ parameters. In practice, often $k = 2$ (binary classification) and $r = 1$ (Bernoulli variables as features) are common, and so the total number of parameters of the naive Bayes model is $2n + 1$, where n is the number of binary features used for classification.

2.2.3 MARKOV RANDOM FIELD

A Markov random field, Markov network or undirected graphical model is set of random variables having a Markov property described by an undirected graph. A Markov random field is similar to a Bayesian Network in its representation of dependencies; the differences being that Bayesian networks are directed and acyclic, whereas Markov networks are undirected and may be cyclic. Thus, a Markov network can represent certain dependencies that a Bayesian network cannot (such as cyclic dependencies); on the other hand, it can't represent certain dependencies that a Bayesian network can (such as induced dependencies).

When the probability distribution is strictly positive, it is also referred to as a Gibbs random field, because, according to the Hammersley–Clifford theorem, it can then be represented by a Gibbs measure. The prototypical Markov random field is the Ising model; indeed, the Markov random field was introduced as the general setting for the Ising model. In the domain of artificial intelligence, a Markov random field is used to model various low- to mid-level tasks in image processing and computer vision. For example, MRFs are used for image restoration, image completion, segmentation, image registration, texture synthesis, super-resolution, stereo matching and Information Retrieval.

Given an undirected graph $G = (V, E)$, a set of random variables $X = (X_v)_{v \in V}$ indexed by V form a Markov random field with respect to G if they satisfy the local Markov properties:

Pairwise Markov property: Any two non-adjacent variables are conditionally independent given all other variables:

$$X_u \perp\!\!\!\perp X_v \mid X_{V \setminus \{u,v\}} \quad \text{if } \{u,v\} \notin E$$

Local Markov property: A variable is conditionally independent of all other variables given its neighbors:

$$X_v \perp\!\!\!\perp X_{V \setminus \mathrm{cl}(v)} \mid X_{\mathrm{ne}(v)}$$

where ne(v) is the set of neighbors of v, and cl(v) = {v} ∪ ne(v) is the closed neighbourhood of v.

Global Markov property: Any two subsets of variables are conditionally independent given a separating subset:

$$X_A \perp\!\!\!\perp X_B \mid X_S$$

where every path from a node in A to a node in B passes through S.

2.3 SOFTWARE DEFECT

2.3.1 WHAT IS A DEFECT OR BUG?

While testing when a tester executes the test cases he might observe that the actual test results do not match from the expected results. The variation in the expected and actual results is known as defects. Different organizations have different names to describe this variation, commonly defects are also known as bug, problem, incidents or issues.

Every incident that occurs during testing may not be a defect or bug. An incident is any situation in which the software system has a questionable behavior, however we call the incident a defect or bug only if the Root Cause is the problem in the tested component.

Incidents can also occur by some other factors as well like testers mistake in test setup, environment error, invalid expected results etc.

2.3.2 SOFTWARE DEFECT

A software bug is an error, flaw, failure, or fault in a computer program or system that produces an incorrect or unexpected result, or causes it to behave in unintended ways. Most bugs arise from mistakes and errors made by people in either a program's source code or its design, and a few are caused by compilers producing incorrect code. A program that contains a large number of bugs, and/or bugs that seriously interfere with its functionality, is said to be buggy. Reports detailing bugs in a program are commonly known as bug reports, defect reports, fault reports, problem reports, trouble reports, change requests, and so forth.

Bugs trigger errors that can in turn have a wide variety of ripple effects, with varying levels of inconvenience to the user of the program. Some bugs have only a subtle effect on the program's functionality, and may thus lie undetected for a long time. More serious bugs may cause the program to crash or freeze. Others qualify as security bugs and might for example enable a malicious user to bypass access controls in order to obtain unauthorized privileges.

2.4 NEED FOR DEFECT PREVENTION

Defect prevention is an important activity in any software project. In most software organizations, the project team focuses on defect detection and rework. Thus, defect prevention, often becomes a neglected component. It is therefore advisable to make measures that prevent the defect from being introduced in the product right from early stages of the project. While the cost of such measures are the minimal, the benefits derived due to overall cost saving are significantly higher compared to cost of fixing the defect at later stage. Thus analysis of the defects at early stages reduces the time, cost and the resources required. The knowledge of defect injecting methods and processes enable the defect prevention. Once this knowledge is practiced the quality is improved. It also enhances the total productivity.

2.5 PREVIOUS WORK

Several methods were used for prediction of software defects. Regression and machine learning methods (decision tree and neural networks) to see the importance of object oriented metrics for fault proneness prediction. They formulate a hypothesis for each object oriented metric and test the correctness of these hypotheses using open source web and email tool Mozilla. For comparison they use precision, correctness and completeness. They find that CBO is the best predictor and LOC is the second. On the other hand, the prediction capability of WMC and RFC is less than CBO and LOC but much better than LCOM, DIT, and NOC. According to the results, DIT is untrustworthy and NOC cannot be used for fault proneness prediction. Furthermore, the correctness of LCOM is good although it has a low completeness value.

After reviewing different techniques for software defect prediction, we can conclude that traditional statistical approaches like regression alone is not enough. Instead they believe that causal models are needed for more accurate predictions. Thus, we propose to use Bayesian networks to predict software defects and software reliability and conclude that using dynamic discretization algorithms while generating Bayesian networks leads to significantly improved accuracy for defects and reliability prediction.

2.6 PROPOSED APPROACH

2.6.1 BAYESIAN NETWORK OF METRICS AND DEFECT PRONENESS

It is very important to model the associational relationships among the metrics and defect proneness. We first generate a Bayesian network among software metrics and defect proneness and then using this network, we calculate an overall marginal defectiveness probability of the software system. This network provides us two very important results:

– The dependencies among the metrics we choose. Which metrics are affected by other metrics and which ones are the most effective on defect proneness.

– The defect proneness probability of the software system itself. By learning from the data set, the Bayesian network tells us the marginal defectiveness probability of the whole system and one can interpret this as the probability of having at least one or more defects in a software module that is selected randomly.

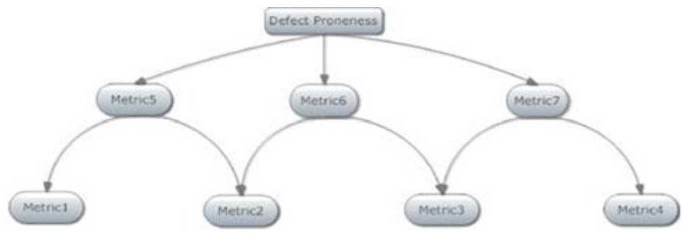


Fig 2.6.1

In this Bayesian network, we see the interactions among different product, process or developer metrics. We may see that a metric is not affected by any other metric whereas some metrics may be affected by one or more product metrics (like $Metric_5$). According to this Bayesian network $Metric_5$, $Metric_6$ and $Metric_7$ are the most important metrics since they affect defectiveness directly. On the other hand, $Metric_1$, $Metric_2$, $Metric_3$ and $Metric_4$ are less important since they are indirectly related with defectiveness.

As a summary, the Bayesian network we propose is a graph G of E edges and V vertices where each V_i represents a metric and each E_j represents the dependency between two metrics or between a metric and defectiveness. If an edge E is present from metric m_2 towards metric m_1 , then this would mean metric m_1 is effective on metric m_2 . Similarly, if there is an edge from defectiveness to metric m_1 , then it would mean that metric m_1 is effective on defectiveness. This way, we determine the metrics that affect defectiveness directly or indirectly.

2.6.2 ORDERING METRICS FOR BAYESIAN NETWORK CONSTRUCTION

In order to learn a Bayesian network with K2 algorithm, it is necessary to specify the order of the nodes. That is why we decide to order the software metrics considering their effect on defectiveness, prior to the generation of Bayesian networks.

We believe that as the size of a software system gets larger, the probability of having fault prone classes increases, since more effort would be needed to ensure a defect free software. We also believe that besides size, complexity of the software

Metric groups	Order (left to right)		
*Group*1	LOC	CBO	LOCQ
*Group*2	WMC	RFC	
*Group*3	LCOM–LCOM3	DIT	NOC

Fig **2.6.2**

is also very important because as the design gets more complex, it would be more difficult for developers to ensure non-defectiveness. That is why, for the initial ordering of the metrics, we decide to give LOC and CBO as the first metrics since LOC is the best indicator of software size and CBO shows how much complex a software system is by counting the number of couples for a certain class where coupling means using methods or instance variables of other classes. As one would easily accept, as coupling increases, the complexity of the software system would also increase. Furthermore, as everybody can accept, when the quality of the source code increase, the probability of having a defective software decreases. So, we introduce the LOCQ metric as the third metric in the first group after LOC and CBO.

Although, RFC may explain complexity to some extent, it may not be the case if a class is using internal methods or instance variables only. That is why, RFC together with WMC are entitled as the second group of metrics. On the other hand, NOC indicates the number of children of a class and is not a good indicator for both size and complexity, since the parent-child relationship does not contribute to the complexity if there is no caller-callee relationship between them which is the case for most of the time. Due to similar reasons DIT also does not explain size or complexity alone. So, we decide to give DIT and NOC as the last metrics in the initial ordering.

Following our reasoning, we generate three groups of metrics where LOC, CBO, and LOCQ are in $Group_1$, WMC and RFC are in $Group_2$ and LCOM, DIT, and NOC are in $Group_3$. $Group_1$ metrics are more important than $Group_2$ metrics and $Group_2$ metrics are more important than $Group_3$ metrics in terms of their effect on defectiveness.

CHAPTER 3

ANALYSIS

3.1 INTRODUCTION

After analyzing the requirements of the task to be performed, the next step is to analyze the problem and understand its context. The first activity in the phase is studying the existing system and other is to understand the requirements and domain of the new system. Both the activities are equally important but the first activity serves as a basis of giving the functional specifications and then successful design of the proposed system. Understanding the properties and requirements of a new system is more difficult and requires creative thinking as well as understanding of existing system is also difficult. Improper understanding of present system can lead diversion from solution.

3.2 BAYESIAN BELIEF NETWORK

Bayes network, belief network, Bayesian model or probabilistic directed acyclic graphical model is a probabilistic graphical model that represents a set of random variables and their conditional dependencies via a directed acyclic graph(DAG). For example, a Bayesian network could represent the probabilistic relationships between diseases and symptoms. Given symptoms, the network can be used to compute the probabilities of the presence of various diseases.

Formally, Bayesian networks are directed acyclic graphs whose nodes represent random variables in the Bayesian sense: they may be observable quantities, latent variables, unknown parameters or hypotheses. Edges represent conditional dependencies; nodes which are not connected represent variables which are conditionally independent of each other. Each node is associated with a probability function that takes as input a particular set of values for the node's parent variables and gives the probability of the variable represented by the node. For example, if the parents are m Boolean variables then the probability function could be represented by a table of 2^m entries, one entry for each of the 2^m possible combinations of its parents being true or false. Similar ideas may be applied to undirected, and possibly cyclic, graphs; such are called Markov networks.

Efficient algorithms exist that perform inference and learning in Bayesian networks. Bayesian networks that model sequences of variables are called dynamic Bayesian networks. Generalizations of Bayesian networks that can represent and solve decision problems under uncertainty are called influence diagrams.

Figure 3.2 shows a sample Bayesian network and conditional probability tables. Assume that we would like to investigate the effect of using experienced developers (ED) and applying unit testing methodology (UT) on defectiveness (FP). Furthermore, each variable can take discrete values of on/off, that is developers are experienced or not, unit testing used or not used. Suppose we would like to make a causal inference and calculate the probability of having a fault prone software if we know that the developers working on the project are experienced. We shall calculate

$$P(FP \mid ED) = P(FP \mid ED, UT)\,P(UT \mid ED) + P(FP \mid ED, \sim UT)\,P(\sim UT \mid ED)$$

We can write $P(UT \mid ED) = P(UT)$ and $P(\sim UT \mid ED) = P(\sim UT)$ since the variables ED and UT are independent. Then we have,

$$P(FP \mid ED) = P(FP \mid ED, UT)\,P(UT) + P(FP \mid ED, \sim UT)\,P(\sim UT)$$

Feeding up the values in the conditional probability table, $P(FP \mid ED)$ is calculated as 0.34. Assume that we are asked to calculate the probability of having experienced developers given the software is fault prone, i.e.

$P(ED \mid FP)$.

Using Bayes' rule we write,

$$P(ED \mid FP) = \frac{P(FP \mid ED)\,P(ED)}{P(FP)}$$

We can also Write

$$P(FP) = P(FP \mid UT, ED)\,P(UT)\,P(ED)$$

$$= P(FP \mid UT, \sim ED)\,P(UT)\,P(\sim ED)$$

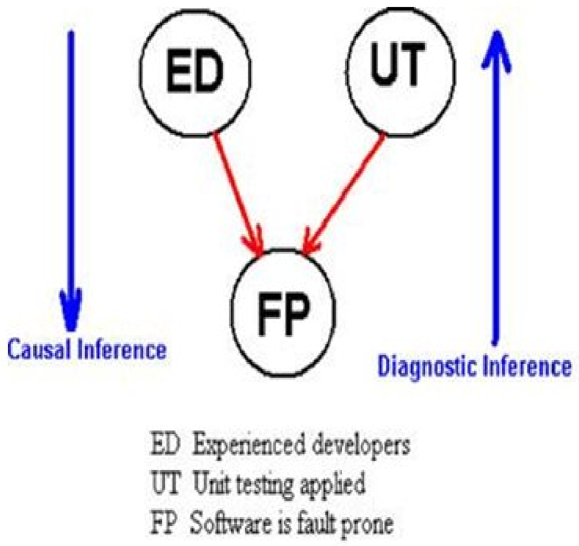
$$= P(FP|\sim UT, ED)\, P(\sim UT)\, P(ED)$$

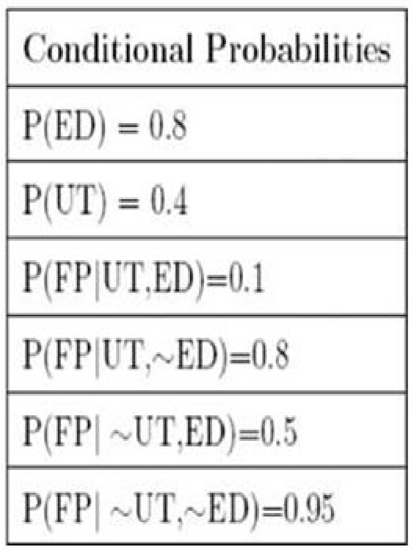
$$+ P(FP|\sim UT, \sim ED)\, P(\sim UT)\, P(\sim ED)$$

Since $P(FP|UT, ED)$, $P(FP|UT, \sim ED)$, $P(FP|\sim UT, ED)$, and

$P(FP|\sim UT, \sim ED)$

can be read from the conditional table, the diagnosis probability $P(ED|FP)$ can also be calculated. As it can be seen in these examples of causal and diagnostic inferences, it is possible to propagate the effect of states of variables (nodes) to calculate posterior probabilities. Propagating the effects of variables to the successors, or analyzing the probability of some predecessor variable based on the probability of its successor is very important in defect prediction since software metrics are related to each other and that is why the weight of a metric might be dependent on another metric based on this relationship.





Conditional Probabilities
P(ED) = 0.8
P(UT) = 0.4
P(FP|UT,ED)=0.1
P(FP|UT,~ED)=0.8
P(FP| ~UT,ED)=0.5
P(FP| ~UT,~ED)=0.95

Fig 3.2

3.3 SOFTWARE REQUIREMENT SPECIFICATION

Purpose: The main purpose for preparing this document is to give a general insight into the analysis and requirements of the existing system or situation and for determining the operating characteristics of the system.

3.3.1 USER REQUIREMENTS

- Knowledge of Bayesian Belief Networks.
- Knowledge of Software Defects.

3.3.2 SOFTWARE REQUIREMENTS

- Operating System : Windows XP/Higher
- BayesiaLab tool.
- Data Sets from data repositories.

3.3.3 HARDWARE REQUIREMENTS

- Processor : Pentium IV
- **Fig 2.6.2**
- Clock speed : 550MHz
- Hard Disk : 20 GB
- RAM : 2 GB
- Cache Memory : 512KB

3.4 CONCLUSION

In this phase, we understand the software requirement specifications for the project. We arrange all the required components to develop the project in this phase itself so that we will have a clear idea regarding the requirements before designing the project. Thus we will proceed to the design phase followed by the implementation phase of the project.

CHAPTER 4

DESIGN

4.1 UML DIAGRAMS

UML is a standard language for specifying, visualizing, constructing, and documenting the artifacts of software systems.

UML was created by Object Management Group (OMG) and UML 1.0 specification draft was proposed to the OMG in January 1997.

OMG is continuously putting effort to make a truly industry standard.

- UML stands for Unified Modeling Language.
- UML is different from the other common programming languages like C++, Java, COBOL etc.
- UML is a pictorial language used to make software blue prints.

So UML can be described as a general purpose visual modeling language to visualize, specify, construct and document software system. Although UML is generally used to model software systems but it is not limited within this boundary. It is also used to model non software systems as well like process flow in a manufacturing unit etc.

UML is not a programming language but tools can be used to generate code in various languages using UML diagrams. UML has a direct relation with object oriented analysis and design. After some standardization UML is become an OMG (Object Management Group) standard.

4.2 GOALS OF UML DIAGRAMS

A picture is worth a thousand words, this absolutely fits while discussing about UML. Object oriented concepts were introduced much earlier than UML. So at that time there were no standard methodologies to organize and consolidate the object oriented development. At that point of time UML came into picture.

There are a number of goals for developing UML but the most important is to define some general purpose modeling language which all modelers can use and also it needs to be made simple to understand and use.

UML diagrams are not only made for developers but also for business users, common people and anybody interested to understand the system. The system can be a software or non software. So it must be clear that UML is not a development method rather it accompanies with processes to make a successful system.

At the conclusion the goal of UML can be defined as a simple modeling mechanism to model all possible practical systems in today.s complex environment.

4.3 A CONCEPTUAL MODEL OF UML

To understand conceptual model of UML first we need to clarify, What is a conceptual model? And, Why a conceptual model is at all required?

- A conceptual model can be defined as a model which is made of concepts and their relationships.
- A conceptual model is the first step before drawing a UML diagram. It helps to understand the entities in the real world and how they interact with each other.

As UML describes the real time systems it is very important to make a conceptual model and then proceed gradually. Conceptual model of UML can be mastered by learning the following three major elements:

- UML building blocks
- Rules to connect the building blocks
- Common mechanisms of UML

4.4 OBJECT ORIENTED CONCEPTS

UML can be described as the successor of object oriented analysis and design.

An object contains both data and methods that control the data. The data represents the state of the object. A class describes an object and they also form hierarchy to model real world system. The hierarchy is represented as inheritance and the classes can also be associated in different manners as per the requirement.

The objects are the real world entities that exist around us and the basic concepts l ike abstraction, encapsulation, inheritance, polymorphism all can be represented using UML.

So UML is powerful enough to represent all the concepts exists in object oriented analysis and design. UML diagrams are representation of object oriented concepts only. So before learning UML, it becomes important to understand OO concepts in details.

Following are some fundamental concepts of object oriented world:

- **Objects:** Objects represent an entity and the basic building block.
- **Class:** Class is the blue print of an object.
- **Abstraction:** Abstraction represents the behavior of an real world entity.
- **Encapsulation:** Encapsulation is the mechanism of binding the data together and hiding them from outside world.
- **Inheritance:** Inheritance is the mechanism of making new classes from existing one.
- **Polymorphism:** It defines the mechanism to exists in different forms.

4.5 OO ANALYSIS AND DESIGN

Object Oriented analysis can be defined as investigation and to be more specific it is the investigation of objects. Design means collaboration of identified objects.

So it is important to understand the OO analysis and design concepts. Now the most important purpose of OO analysis is to identify objects of a system to be designed. This analysis is also done for an existing system. Now an efficient analysis is only possible when we are able to start thinking in a way where objects can be identified. After identifying the objects their relationships are identified and finally the design is produced.

So the purpose of OO analysis and design can described as:

- Identifying the objects of a system.
- Identify their relationships.
- Make a design which can be converted to executables using OO languages.

There are three basic steps where the OO concepts are applied and implemented. The

```
OO Analysis --> OO Design --> OO implementation using OO languages
```

steps can be defined as

Now the above three points can be described in details:

- During object oriented analysis the most important purpose is to identify objects and describing them in a proper way. If these objects are identified efficiently then the next job of design is easy. The objects should be identified with responsibilities. Responsibilities are the functions performed by the object. Each and every object has some type of responsibilities to be performed. When these responsibilities are collaborated the purpose of the system is fulfilled.
- The second phase is object oriented design. During this phase emphasis is given upon the requirements and their fulfilment. In this stage the objects are collaborated according to their intended association. After the association is complete the design is also complete.
- The third phase is object oriented implementation. In this phase the design is implemented using object oriented languages like Java, C++ etc.

4.6 ROLE OF UML IN OO DESIGN

UML is a modeling language used to model software and non software systems. Although UML is used for non software systems the emphasis is on modeling object oriented software applications. Most of the UML diagrams discussed so far are used to model different aspects like static, dynamic etc. Now what ever be the aspect the artifacts are nothing but objects.

If we look into class diagram, object diagram, collaboration diagram, interaction diagrams all would basically be designed based on the objects.

So the relation between OO design and UML is very important to understand. The OO design is transformed into UML diagrams according to the requirement. Before understanding the UML in details the OO concepts should be learned properly. Once the OO analysis and design is done the next step is very easy. The input from the OO analysis and design is the input to the UML diagrams.

4.7 UML ARCHITECTURE

Any real world system is used by different users. The users can be developers, testers, business people, analysts and many more. So before designing a system the architecture is made with different perspectives in mind. The most important part is to visualize the system from different viewer.s perspective. The better we understand the better we make the system.

UML plays an important role in defining different perspectives of a system. These perspectives are:

- Design
- Implementation
- Process
- Deployment

And the centre is the Use Case view which connects all these four. A Use case represents the functionality of the system. So the other perspectives are connected with use case.

- **Design** of a system consists of classes, interfaces and collaboration. UML provides class diagram, object diagram to support this.
- **Implementation** defines the components assembled together to make a complete physical system. UML component diagram is used to support implementation perspective.
- **Process** defines the flow of the system. So the same elements as used in Design are also used to support this perspective.
- **Deployment** represents the physical nodes of the system that forms the hardware. UML deployment diagram is used to support this perspective.

4.8 UML MODELLING TYPES

It is very important to distinguish between the UML model. Different diagrams are used for different type of UML modeling. There are three important type of UML modelings:

4.8.1 Structural Modeling:

Structural modeling captures the static features of a system. They consist of the followings:

- Classes diagrams
- Objects diagrams
- Deployment diagrams
- Package diagrams
- Composite structure diagram
- Component diagram

Structural model represents the framework for the system and this framework is the place where all other components exist. So the class diagram, component diagram and deployment diagrams are the part of structural modeling. They all represent the elements and the mechanism to assemble them.

But the structural model never describes the dynamic behavior of the system. Class diagram is the most widely used structural diagram.

4.8.2 Behavioral Modeling:

Behavioral model describes the interaction in the system. It represents the interaction among the structural diagrams. Behavioral modeling shows the dynamic nature of the system. They consist of the following:

- Activity diagrams
- Interaction diagrams
- Use case diagrams

4.8.3 Architectural Modeling:

Architectural model represents the overall framework of the system. It contains both structural and behavioral elements of the system. Architectural model can be defined as the blue print of the entire system. Package diagram comes under architectural modeling.

4.9 USE CASE DIAGRAM

Use case diagram is used to capture the dynamic nature of a system. It consists of use cases, actors and their relationships. Use case diagram is used at a high level design to capture the requirements of a system.

So it represents the system functionalities and their flow. Although the use case diagrams are not a good candidate for forward and reverse engineering but still they are used in a slightly differently way to model it.

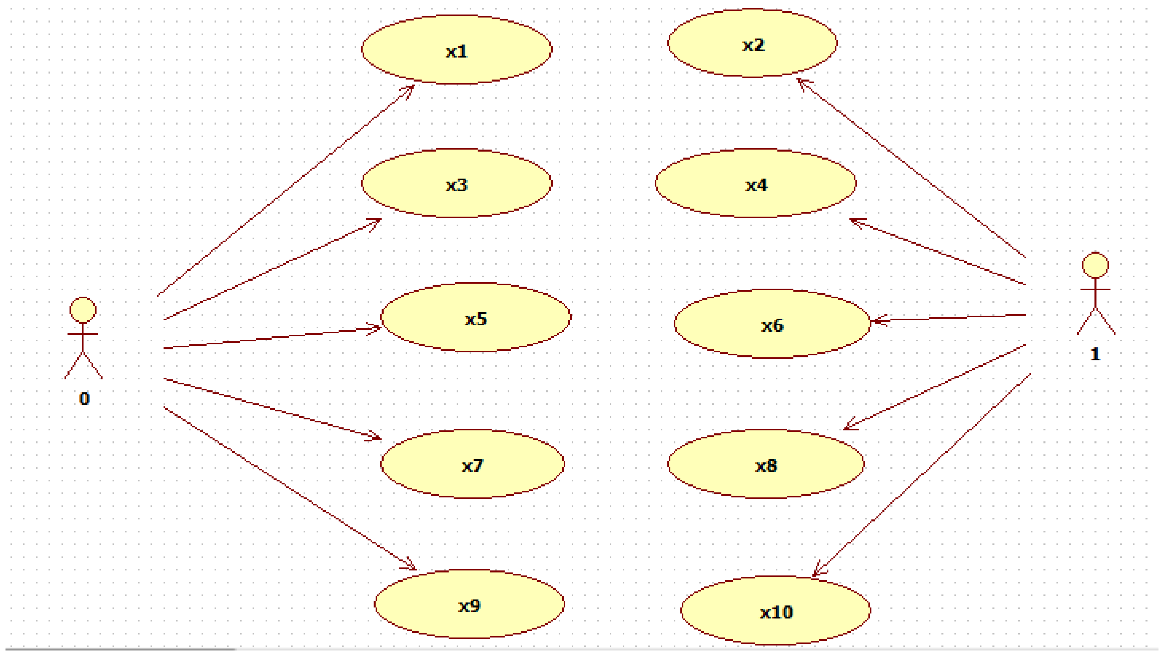


Fig 4.9

CHAPTER 5

IMPLEMENTATION

5.1 BAYESIALAB OVERVIEW

BayesiaLab is a powerful desktop application (Windows/Mac/Unix) for nowledge management, data mining, analytics, predictive modeling and simulation all based on the paradigm of Bayesian networks. Bayesian networks have become a very powerful tool for deep understanding of very complex, high-dimensional problem domains, ranging from bioinformatics to marketing science.

BayesiaLab is the world's only comprehensive software package for generating, manipulating and analyzing Bayesian networks. Analysts and researchers around the world, including Bayesia's strategic partner P&G, have embraced BayesiaLab to gain unprecedented insights into problems which had previously not been tractable with traditional analysis methods.

While cutting-edge research tools are often of no practical use outside the laboratory, BayesiaLab is a major exception. BayesiaLab provides an extremely user-friendly interface that allows novices and experts alike to easily and quickly navigate all the functions available in the program. Intuitive menu structures and step-by-step wizards allow end-users to focus on their principal analysis task without having to worry about idiosyncratic syntax or arcane commands.

5.2 CONFUSION MATRIX

A confusion matrix (Kohavi and Provost, 1998) contains information about actual and predicted classifications done by a classification system. Performance of such systems is commonly evaluated using the data in the matrix. The following table shows the confusion matrix for a two class classifier.

The entries in the confusion matrix have the following meaning in the context of our study:

- *a* is the number of correct predictions that an instance is negative,
- *b* is the number of incorrect predictions that an instance is positive,
- *c* is the number of incorrect of predictions that an instance negative, and
- *d* is the number of correct predictions that an instance is positive.

		Predicted	
		Negative	Positive
Actual	Negative	**A**	**B**
	Positive	**C**	**D**

Several standard terms have been defined for the 2 class matrix:

- The accuracy (*AC*) is the proportion of the total number of predictions that were correct. It is determined using the equation:

$$\text{Ac}=\frac{?+?}{?+?+?+?}$$

- The recall or true positive rate *(TP)* is the proportion of positive cases that were correctly identified, as calculated using the equation:

$$\text{TP}=\frac{?}{?+?}$$

- The false positive rate (FP) is the proportion of negatives cases that were incorrectly classified as positive, as calculated using the equation:

$$\text{FP}=\frac{?}{?+?}$$

- The true negative rate (TN) is defined as the proportion of negatives cases that were classified correctly, as calculated using the equation:

$$TN=\frac{?}{? + ?}$$

- The false negative rate (FN) is the proportion of positives cases that were incorrectly classified as negative, as calculated using the equation:

$$FN=\frac{?}{? + ?}$$

- Finally, precision (P) is the proportion of the predicted positive cases that were correct, as calculated using the equation:

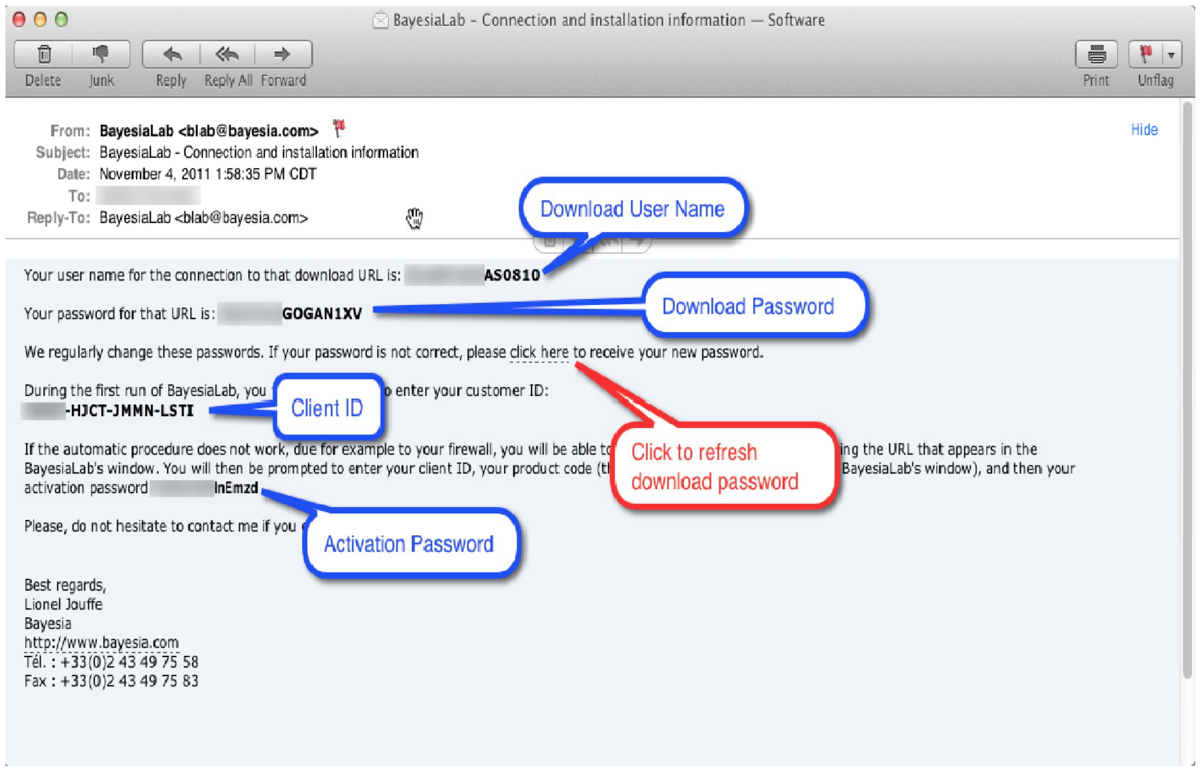
$$P=\frac{?}{? + ?}$$

The accuracy determined using equation 1 may not be an adequate performance measure when the number of negative cases is much greater than the number of positive cases (Kubat et al., 1998). Suppose there are 1000 cases, 995 of which are negative cases and 5 of which are positive cases. If the system classifies them all as negative, the accuracy would be 99.5%, even though the classifier missed all positive cases. Other performance measures account for this by including *TP* in a product

5.3 BAYESIALAB INSTALLATION

5.3.1 ACCOUNT INFORMATION

Before you can start the installation, you will need to receive an email from the Bayesia License Server with your Bayesia account information, which includes multiple credentials:



From: BayesiaLab <blab@bayesia.com>
Subject: BayesiaLab - Connection and installation information
Date: November 4, 2011 1:58:35 PM CDT
To: [redacted]
Reply-To: BayesiaLab <blab@bayesia.com>

Your user name for the connection to that download URL is: [redacted]AS0810

Your password for that URL is: [redacted]GOGAN1XV

We regularly change these passwords. If your password is not correct, please click here to receive your new password.

During the first run of BayesiaLab, you o enter your customer ID:
[redacted]-HJCT-JMMN-LSTI

If the automatic procedure does not work, due for example to your firewall, you will be able t ing the URL that appears in the BayesiaLab's window. You will then be prompted to enter your client ID, your product code (t BayesiaLab's window), and then your activation password [redacted]lnEmzd

Please, do not hesitate to contact me if you

Best regards,
Lionel Jouffe
Bayesia
http://www.bayesia.com
Tél. : +33(0)2 43 49 75 58
Fax : +33(0)2 43 49 75 83

Fig 5.3.1

Download Credentials

User Name for software download

Password for software download

These will be required to log onto the Bayesia server to download the BayesiaLab software. For security reasons, this password changes frequently. In case your password has expired, click on the link in your email to receive a new one (see red callout in the above screenshot).

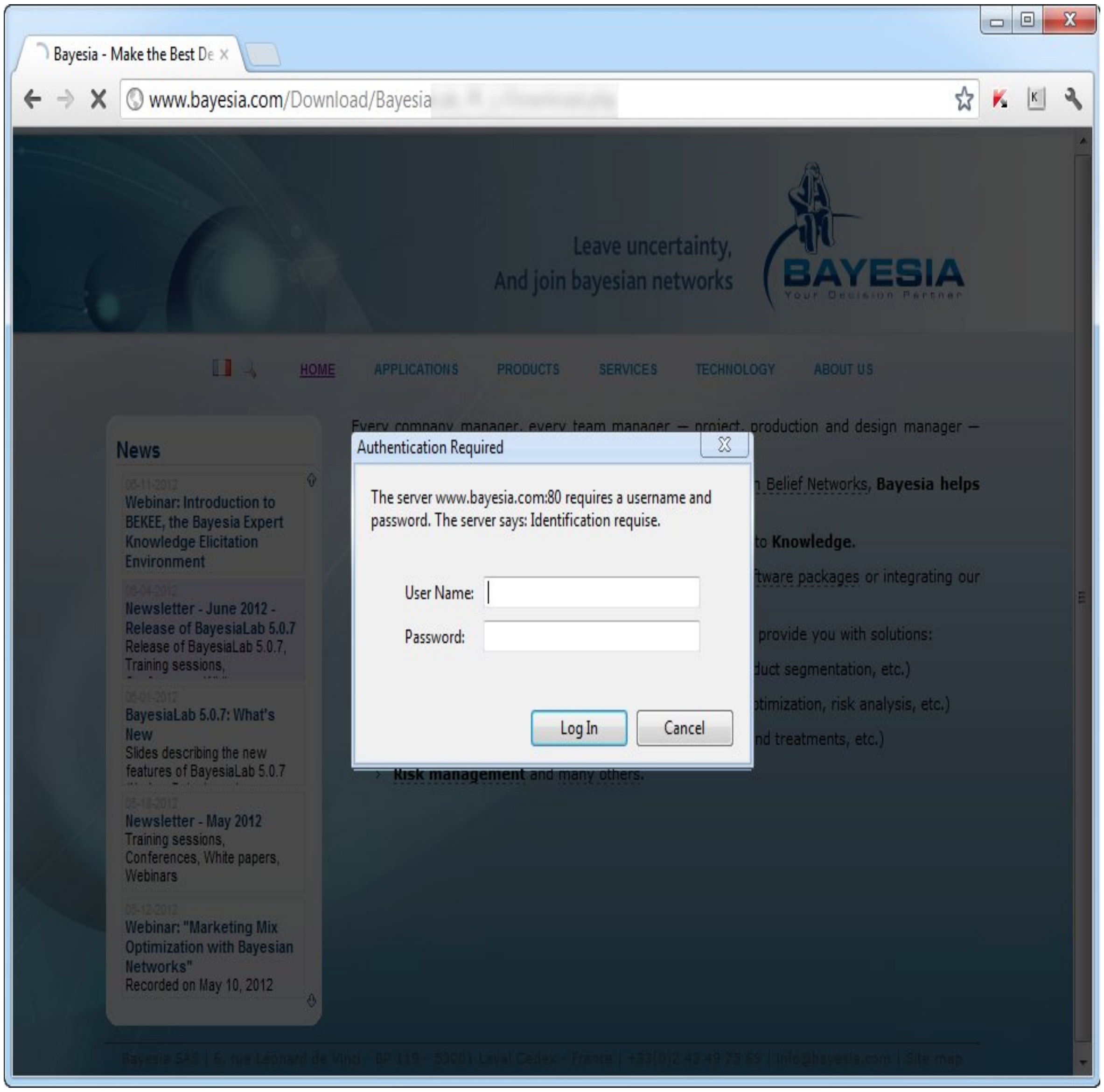
Activation Credentials

Client ID/Customer ID (Format: XXXX-XXXX-XXXX-XXXX)

Activation Password

5.3.2 SOFTWARE DOWNLOAD

We receive a separate email with the URL corresponding to your version of the program, Upon entering this URL, you will be asked to provide your **Download Credentials**, i.e. **User Name** and **Password**



Authentication Required

The server www.bayesia.com:80 requires a username and password. The server says: Identification requise.

User Name:

Password:

Webinar: Introduction to BEKEE, the Bayesia Expert Knowledge Elicitation Environment

Newsletter - June 2012 - Release of BayesiaLab 5.0.7
Release of BayesiaLab 5.0.7, Training sessions,

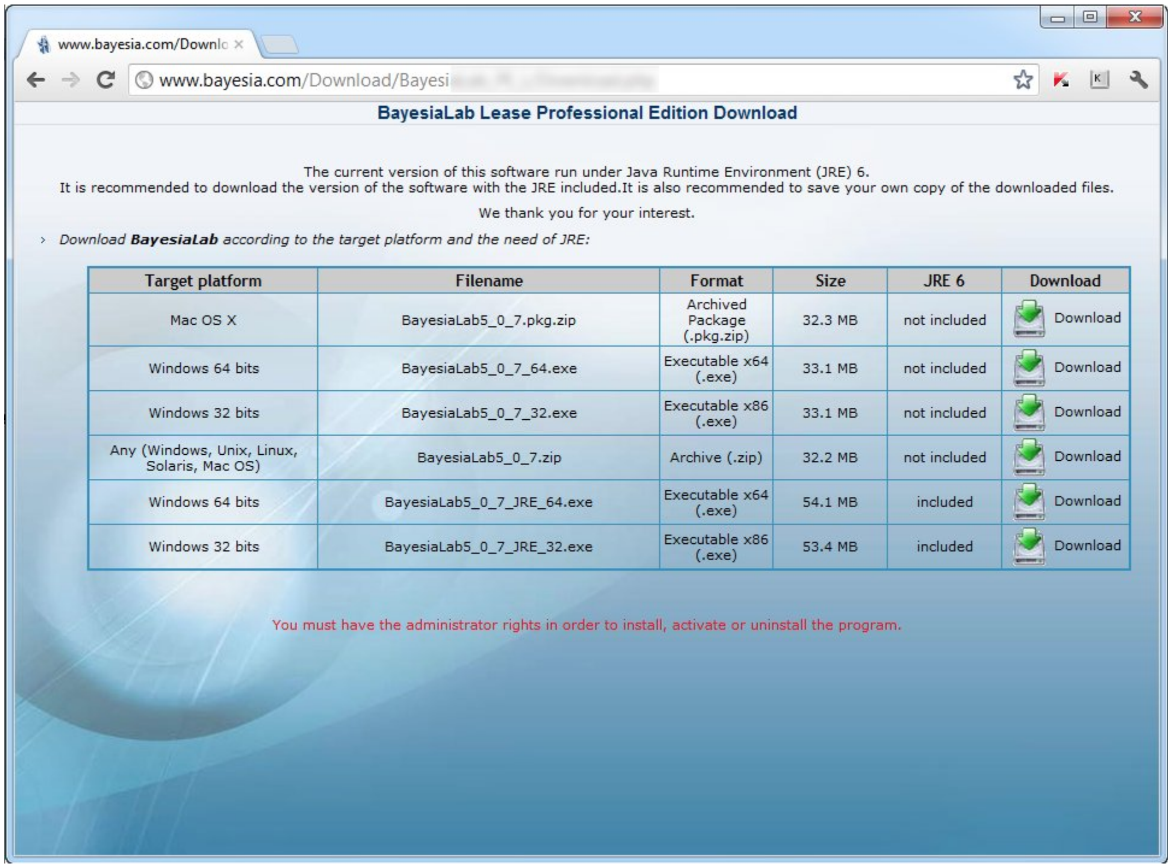
BayesiaLab 5.0.7: What's New
Slides describing the new features of BayesiaLab 5.0.7

Newsletter - May 2012
Training sessions, Conferences, White papers, Webinars

Webinar: "Marketing Mix Optimization with Bayesian Networks"
Recorded on May 10, 2012

Fig 5.3.2

Once you are logged in, you will see several BayesiaLab versions available for download.



BayesiaLab Lease Professional Edition Download

The current version of this software run under Java Runtime Environment (JRE) 6.
It is recommended to download the version of the software with the JRE included.It is also recommended to save your own copy of the downloaded files.
We thank you for your interest.

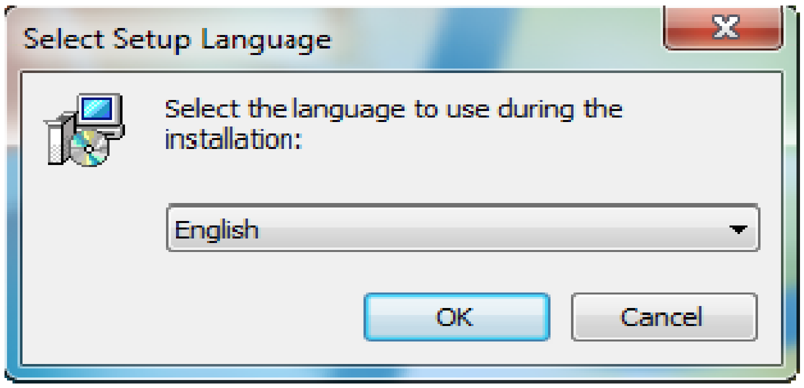
› *Download* ***BayesiaLab*** *according to the target platform and the need of JRE:*

Target platform	Filename	Format	Size	JRE 6	Download
Mac OS X	BayesiaLab5_0_7.pkg.zip	Archived Package (.pkg.zip)	32.3 MB	not included	Download
Windows 64 bits	BayesiaLab5_0_7_64.exe	Executable x64 (.exe)	33.1 MB	not included	Download
Windows 32 bits	BayesiaLab5_0_7_32.exe	Executable x86 (.exe)	33.1 MB	not included	Download
Any (Windows, Unix, Linux, Solaris, Mac OS)	BayesiaLab5_0_7.zip	Archive (.zip)	32.2 MB	not included	Download
Windows 64 bits	BayesiaLab5_0_7_JRE_64.exe	Executable x64 (.exe)	54.1 MB	included	Download
Windows 32 bits	BayesiaLab5_0_7_JRE_32.exe	Executable x86 (.exe)	53.4 MB	included	Download

You must have the administrator rights in order to install, activate or uninstall the program.

Fig 5.3.3

The installer program will first prompt you for the language.



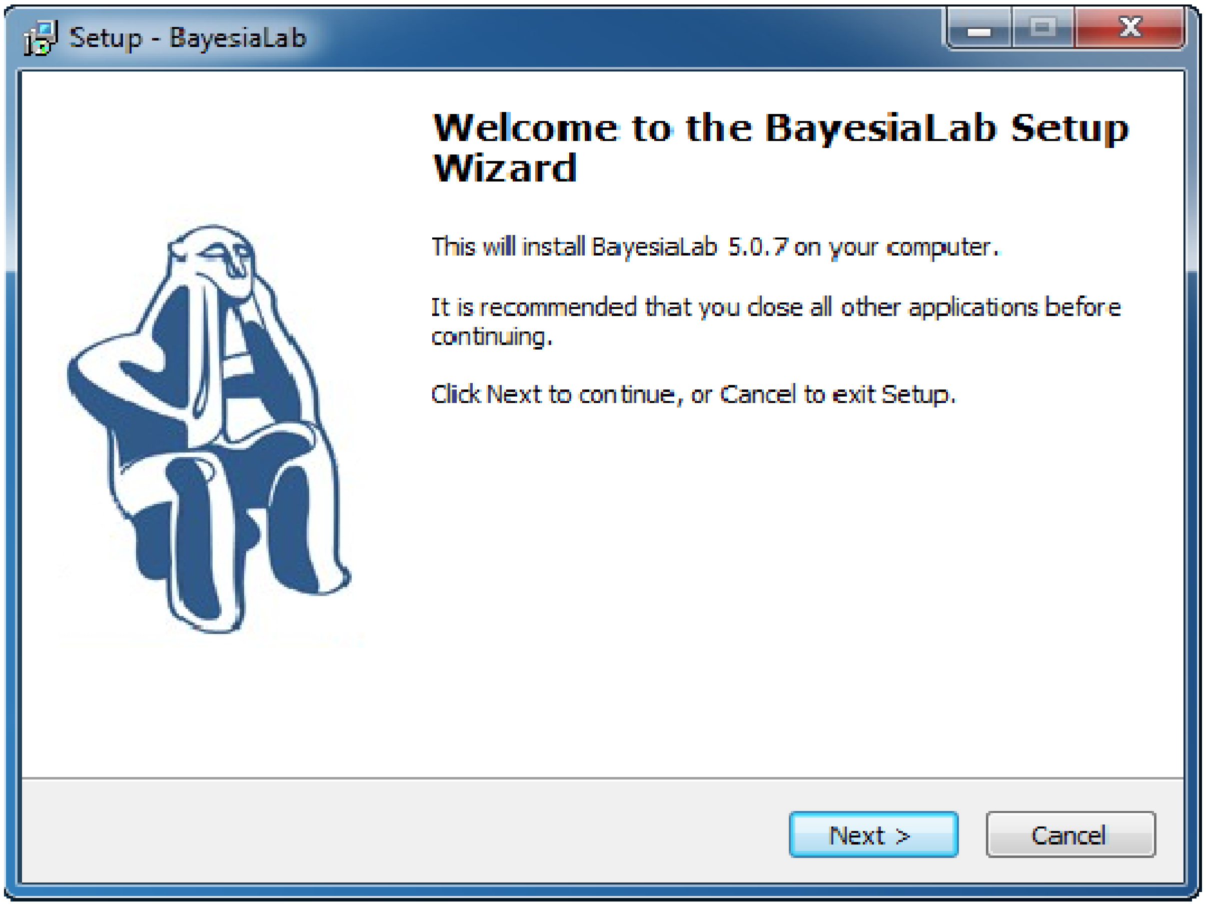
Select Setup Language

Select the language to use during the installation:

English

Fig 5.3.4

Upon language selection, the BayesiaLab Setup Wizard will start:


Setup - BayesiaLab

Welcome to the BayesiaLab Setup Wizard

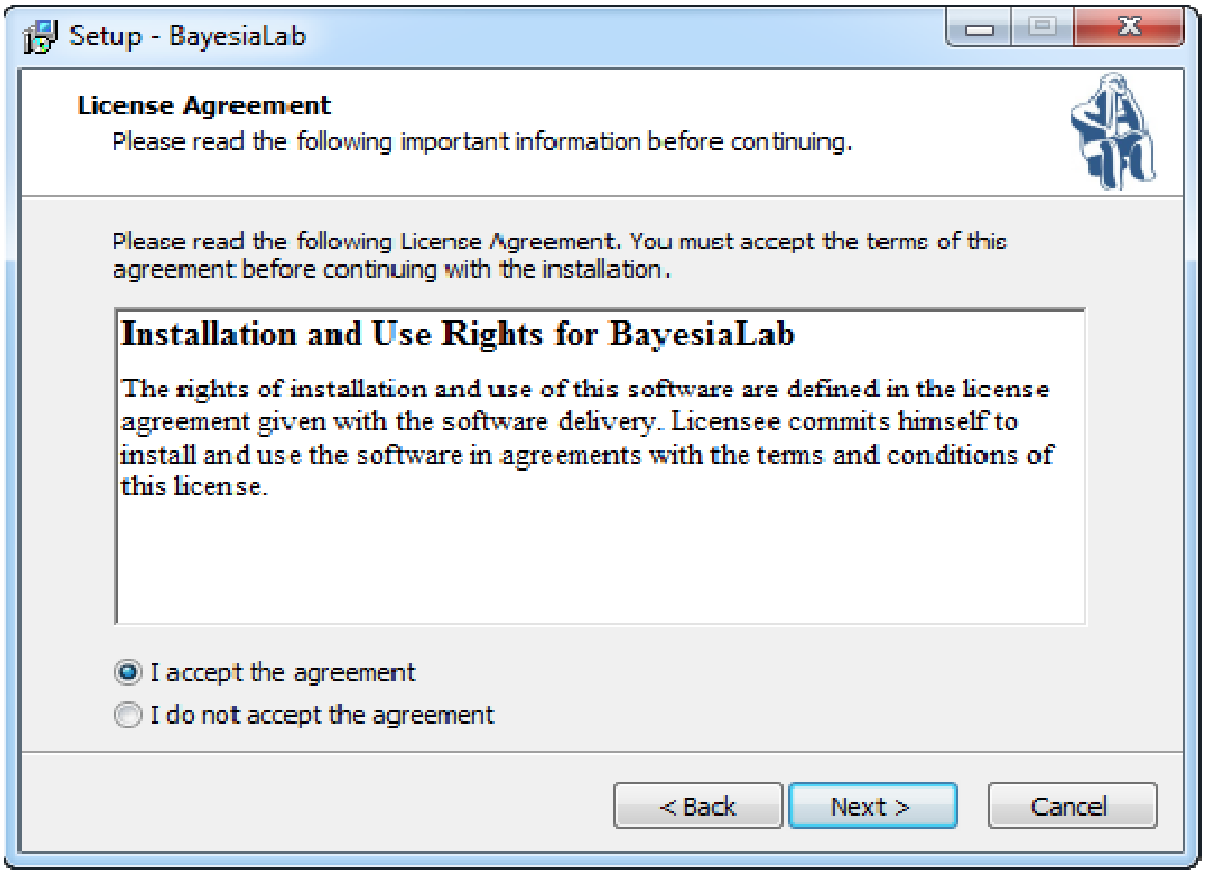
This will install BayesiaLab 5.0.7 on your computer.

It is recommended that you close all other applications before continuing.

Click Next to continue, or Cancel to exit Setup.

Fig 5.3.5

Click next to proceed to the License Agreement.



Setup - BayesiaLab

License Agreement

Please read the following important information before continuing.

Please read the following License Agreement. You must accept the terms of this agreement before continuing with the installation.

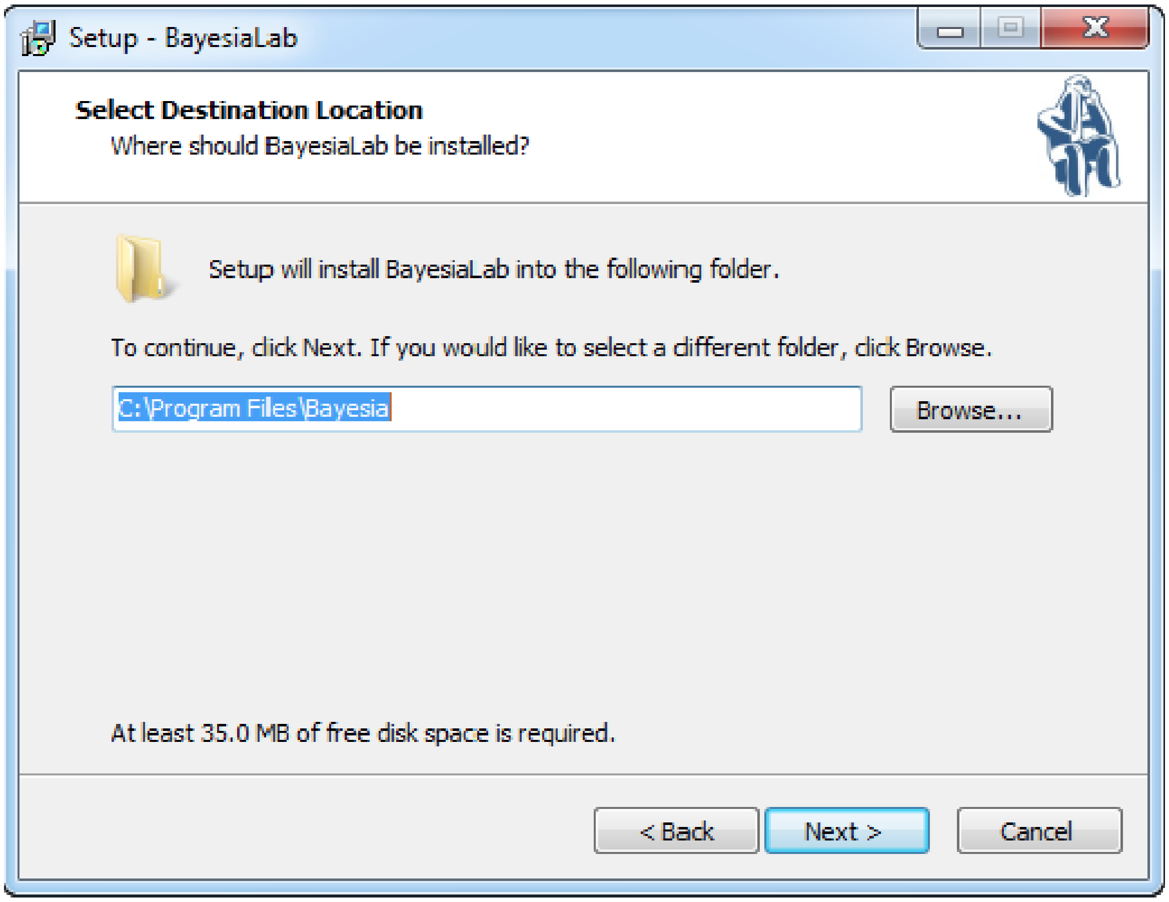
Installation and Use Rights for BayesiaLab

The rights of installation and use of this software are defined in the license agreement given with the software delivery. Licensee commits himself to install and use the software in agreements with the terms and conditions of this license.

[✓ I accept the agreement]
I do not accept the agreement

Fig 5.3.6

After accepting the agreement, the next screen will prompt you to select an installation directory.


Setup - BayesiaLab

Select Destination Location

Where should BayesiaLab be installed?

Setup will install BayesiaLab into the following folder.

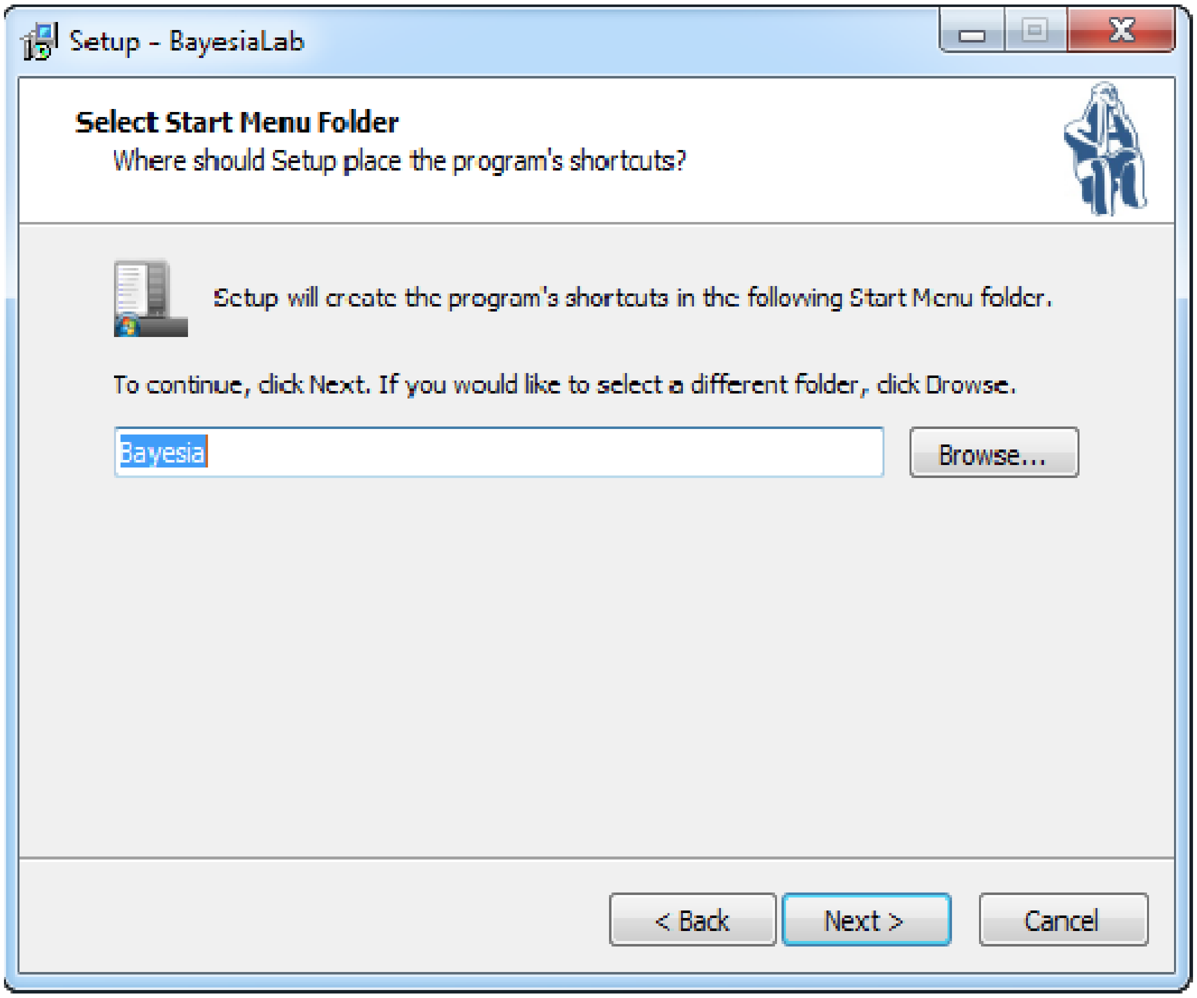
To continue, click Next. If you would like to select a different folder, click Browse.

C:\Program Files\Bayesia

At least 35.0 MB of free disk space is required.

Fig 5.3.7

Windows users can also choose the Start Menu folder, from which BayesiaLab can be started.


Setup - BayesiaLab

Select Start Menu Folder

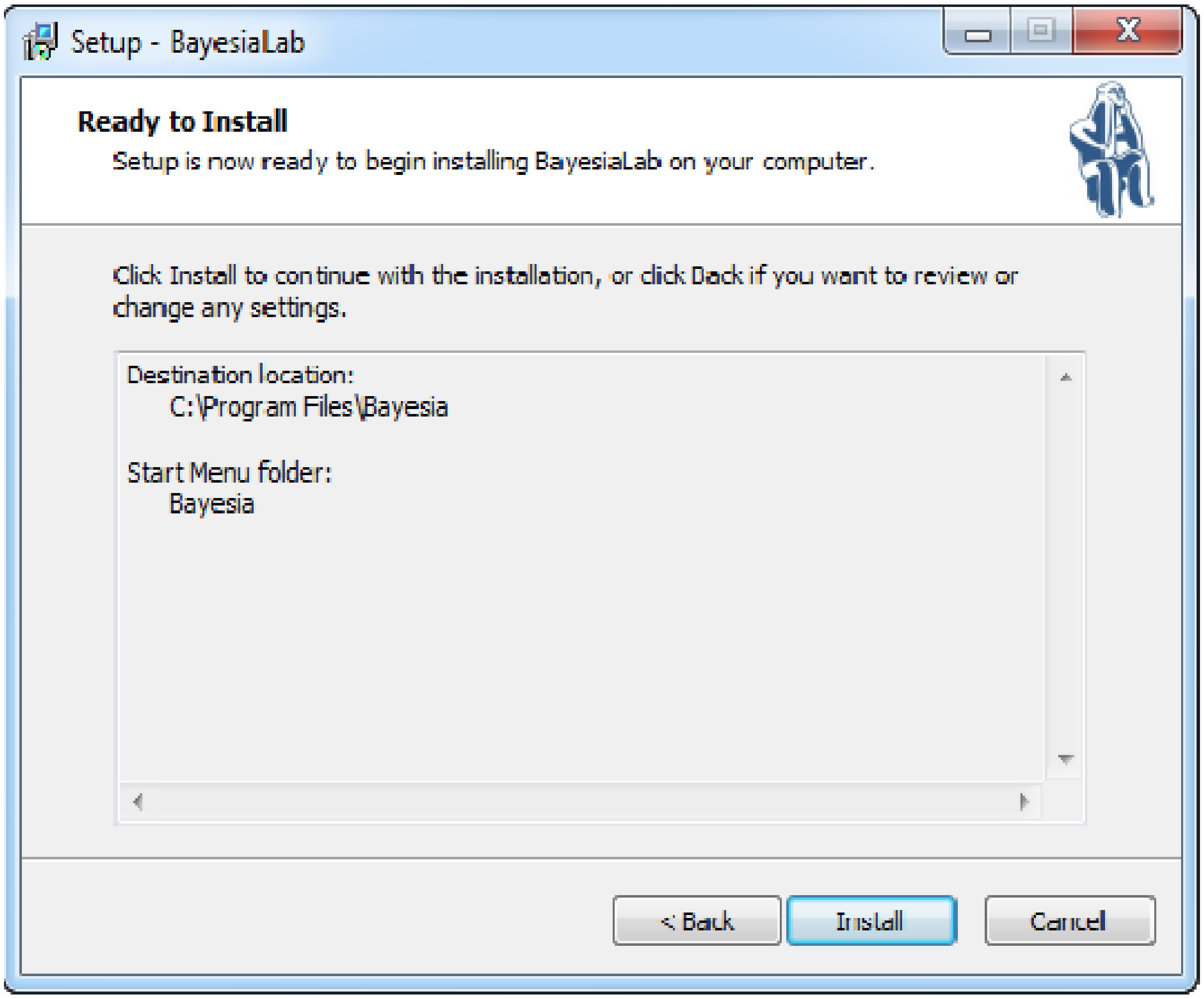
Where should Setup place the program's shortcuts?

Setup will create the program's shortcuts in the following Start Menu folder.

To continue, click Next. If you would like to select a different folder, click Browse.

Bayesia

The next screen confirms the selections and allows you to start the actual installation process.



Ready to Install

Setup is now ready to begin installing BayesiaLab on your computer.

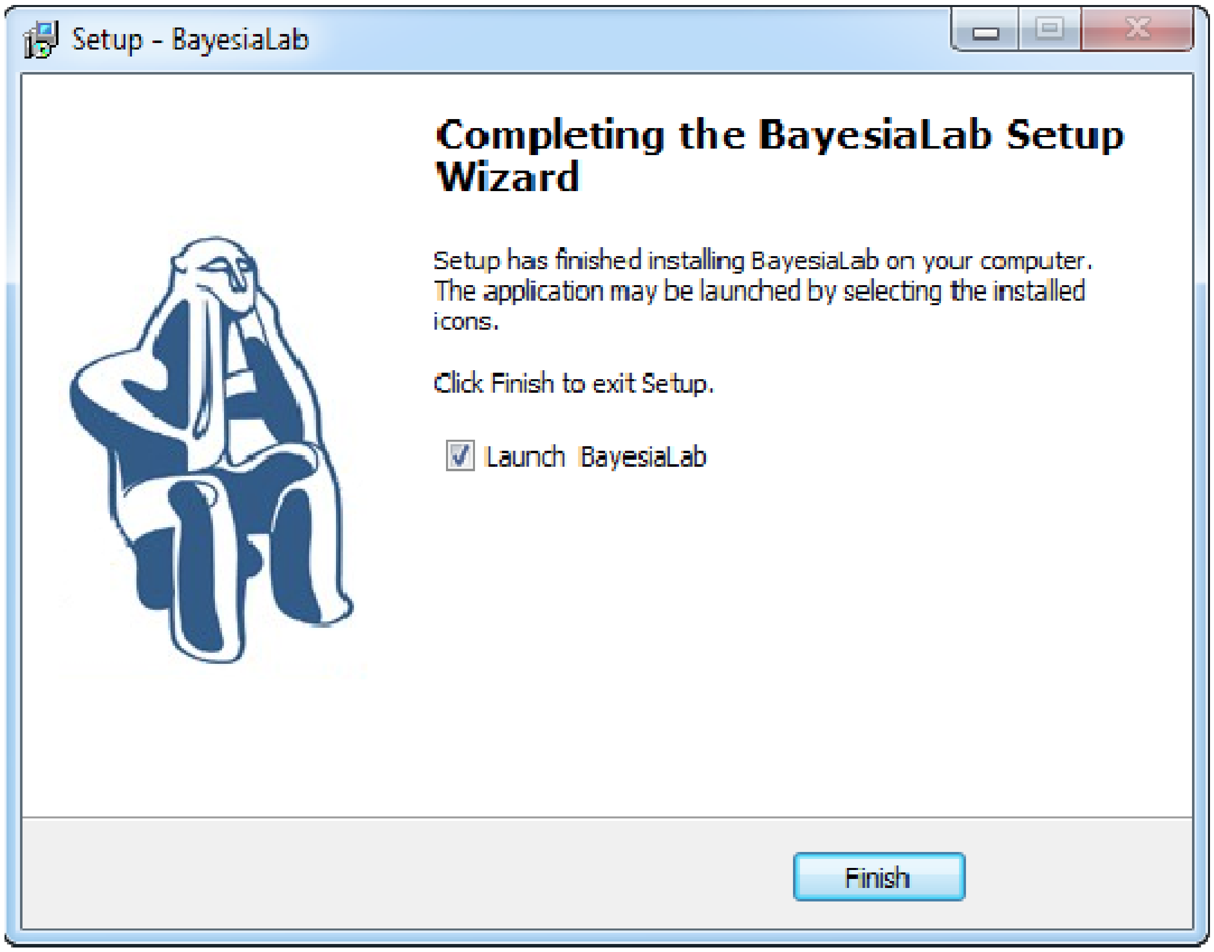
Click Install to continue with the installation, or click Back if you want to review or change any settings.

Destination location:
C:\Program Files\Bayesia

Start Menu folder:
Bayesia

Fig 5.3.8

Upon completion, the Wizard presents this screen.



Completing the BayesiaLab Setup Wizard

Setup has finished installing BayesiaLab on your computer. The application may be launched by selecting the installed icons.

Click Finish to exit Setup.

[✓] Launch BayesiaLab

Fig 5.3.9

This concludes the installation of files, however, you will still need to activate BayesiaLab. To do so, you want to proceed and launch BayesiaLab straight away, while you have all the credentials at hand.

5.4 SCREEN SHOTS

Main Window

BayesiaLab's principal work environment is the Main Window.

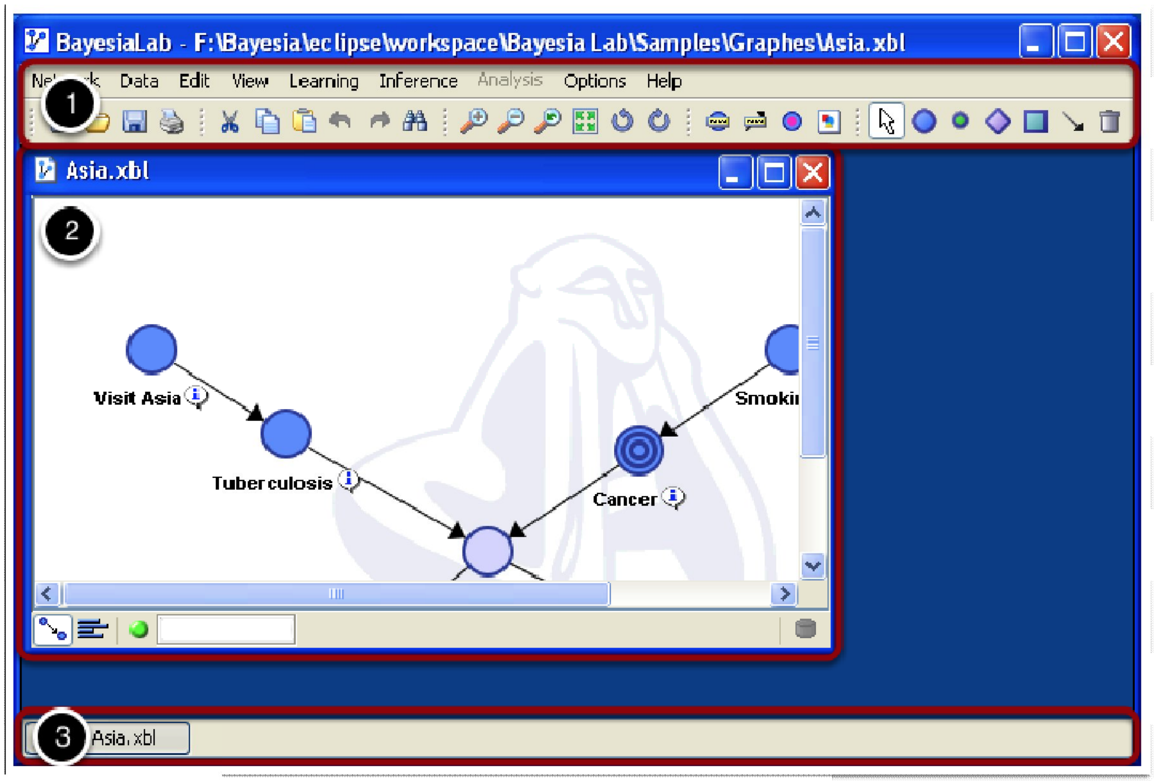

BayesiaLab - F:\Bayesia\eclipse\workspace\Bayesia Lab\Samples\Graphes\Asia.xbl
Network Data Edit View Learning Inference Analysis Options Help
1
Asia.xbl
2
Visit Asia
Tuberculosis
Cancer
Smoki
3
Asia.xbl

Fig 5.4.1

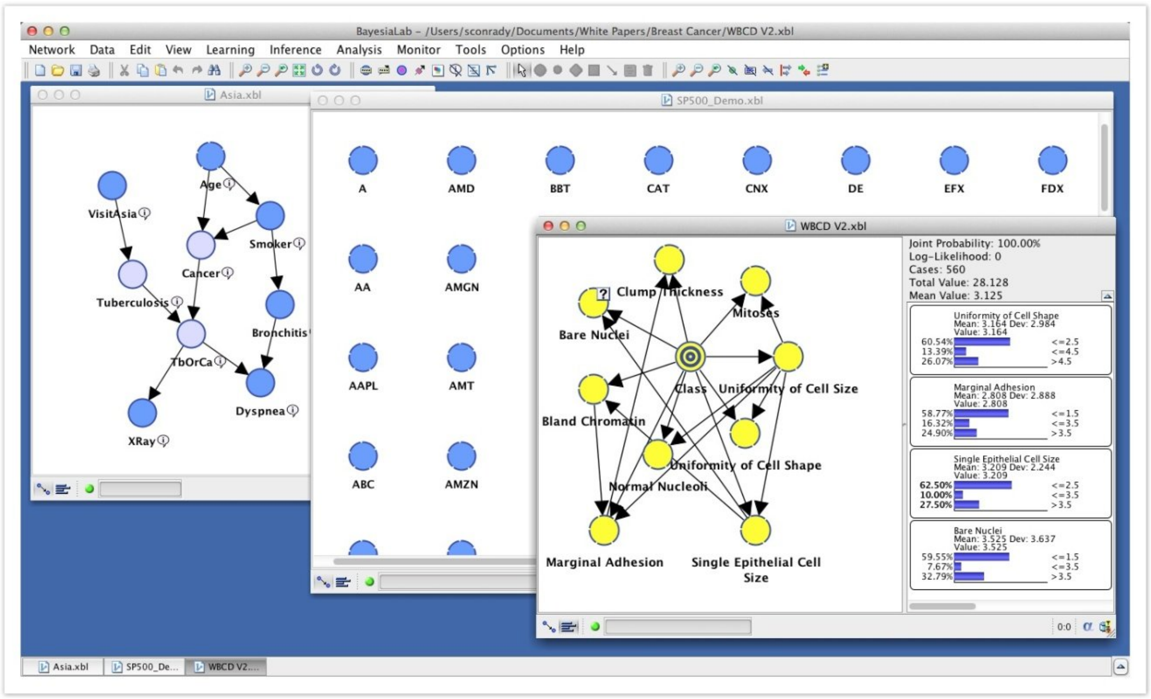
It consists of three major elements:

1. The **Command Zone** (menus and toolbar) that contains all the commands that can operate either on all the graphs or on the active graph.
2. The **Graph Zone** in which graph windows are opened. The **Contextual Menu**, available by right-clicking the **Graph Zone**, allows managing the console display.
3. The **Graph Bar**, at the bottom of the screen, allows managing the graph windows. Similar to selecting the tab of worksheet tab, clicking on the graph button (labeled with the network name) brings up the corresponding graph.

Graph Windows

Each Graph Window represents a distinct entity, consisting of both qualitative and quantitative information.

- Qualitative: i.e. the structure of the graph (Nodes and Arcs),
- Quantitative: i.e. Conditional Probability Tables and, if available, any associated dataset.



Joint Probability: 100.00%
Log-Likelihood: 0
Cases: 560
Total Value: 28.128
Mean Value: 3.125

Uniformity of Cell Shape
Mean: 3.164 Dev: 2.984
Value: 3.164
60.54% <=2.5
13.39% <=4.5
26.07% >4.5

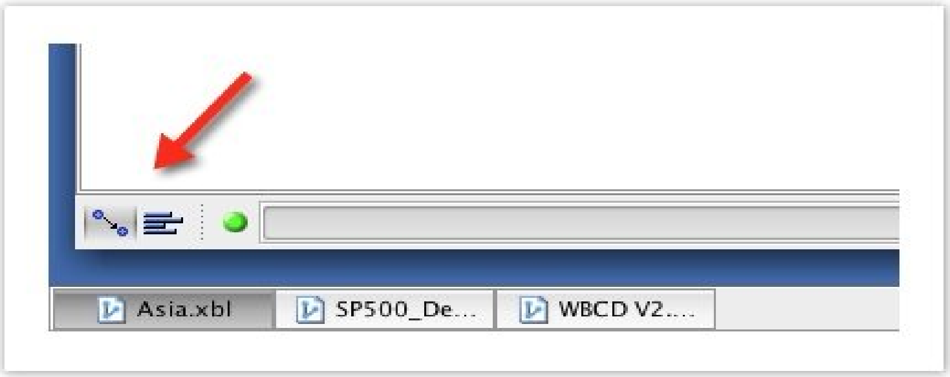
Marginal Adhesion
Mean: 2.808 Dev: 2.888
Value: 2.808
58.77% <=1.5
16.32% <=3.5
24.90% >3.5

Single Epithelial Cell Size
Mean: 3.209 Dev: 2.244
Value: 3.209
62.50% <=2.5
10.00% <=3.5
27.50% >3.5

Bare Nuclei
Mean: 3.525 Dev: 3.637
Value: 3.525
59.55% <=1.5
7.67% <=3.5
32.79% >3.5

Fig 5.4.2

A **Graph Window** can be in two different modes, as indicated by two buttons in the lower lefthand corner of each **Graph Window**



You can switch between modes by clicking on the respective buttons.

- **Modeling Mode**: in this mode all learning and modeling tasks take place. Only the **Graph Panel** is visible.

- **Validation Mode**: in this mode both the **Graph Panel** and the **Monitor Panel** are visible. Model validation, analysis, simulation, etc. are performed exclusively in this mode.

Starting BayesiaLab

Click on BayesiaLab icon after installing the software. BayesiaLab window will be displayed.

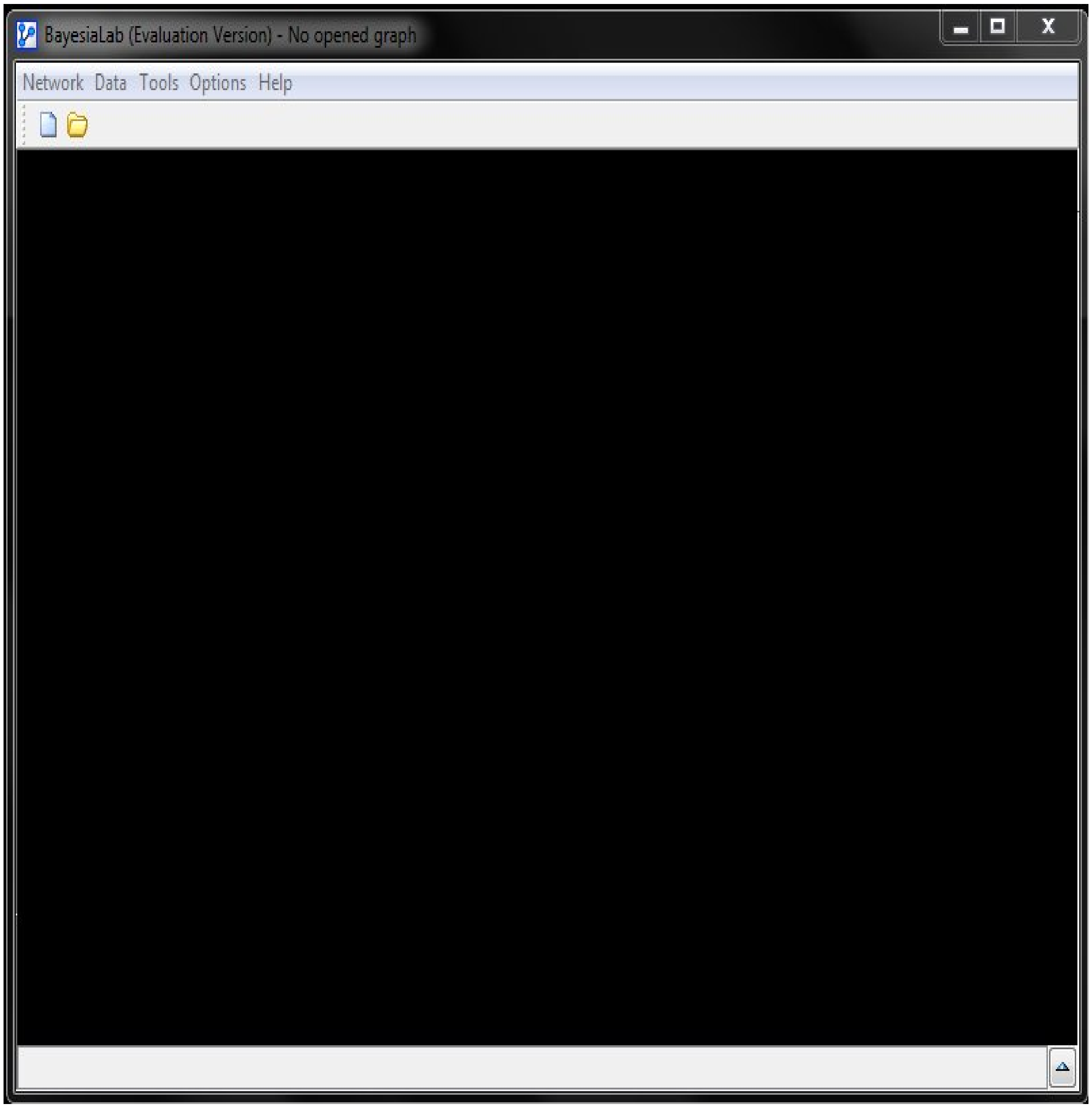

BayesiaLab (Evaluation Version) - No opened graph
Network Data Tools Options Help

Fig 5.4.3

Data Menu

Open the ‘Data’ drop down menu from the main display.

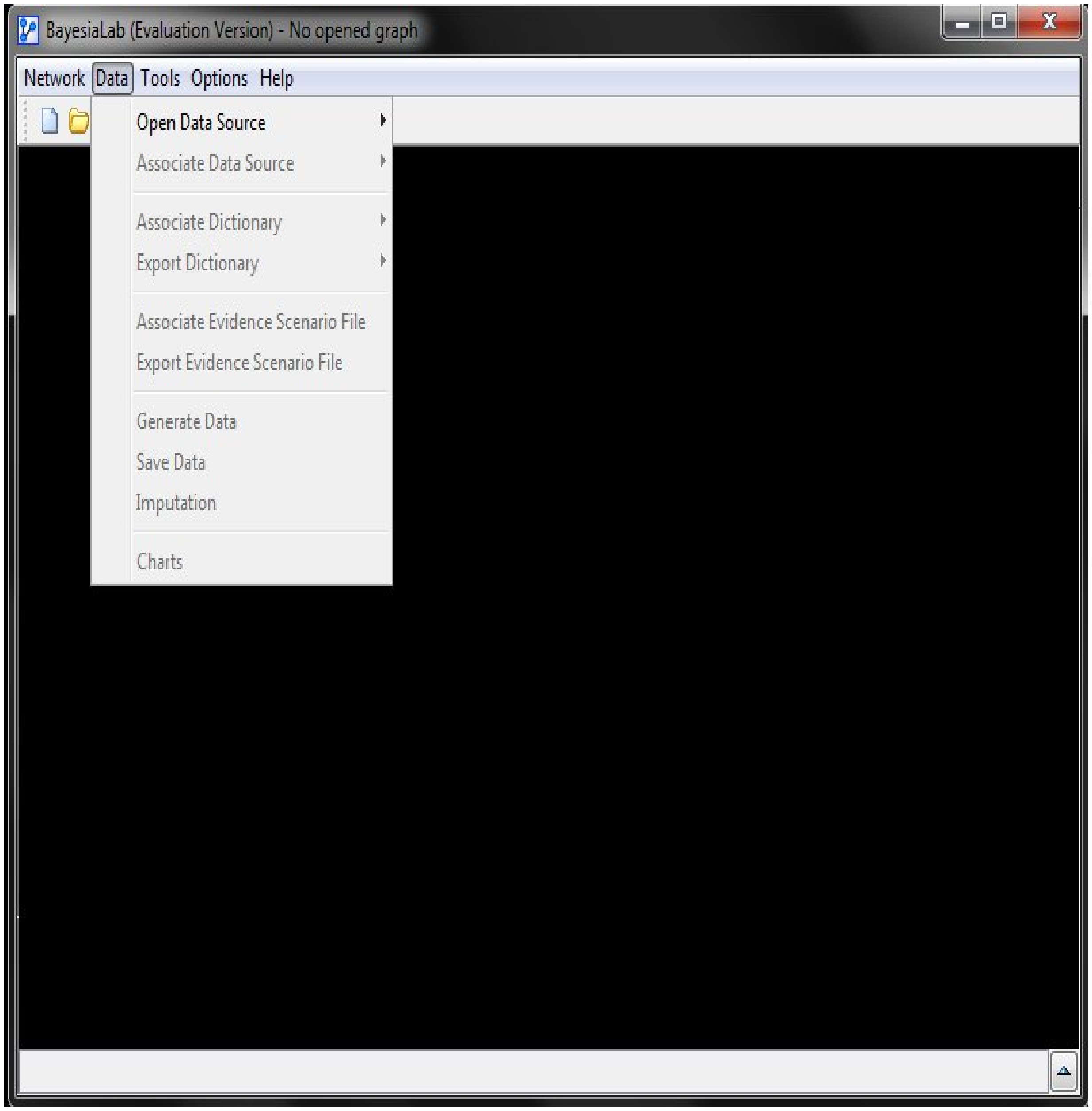


Fig 5.4.4

Text File Selection

Select the Text File option after opening the Open Data Source option in the Data drop down menu.

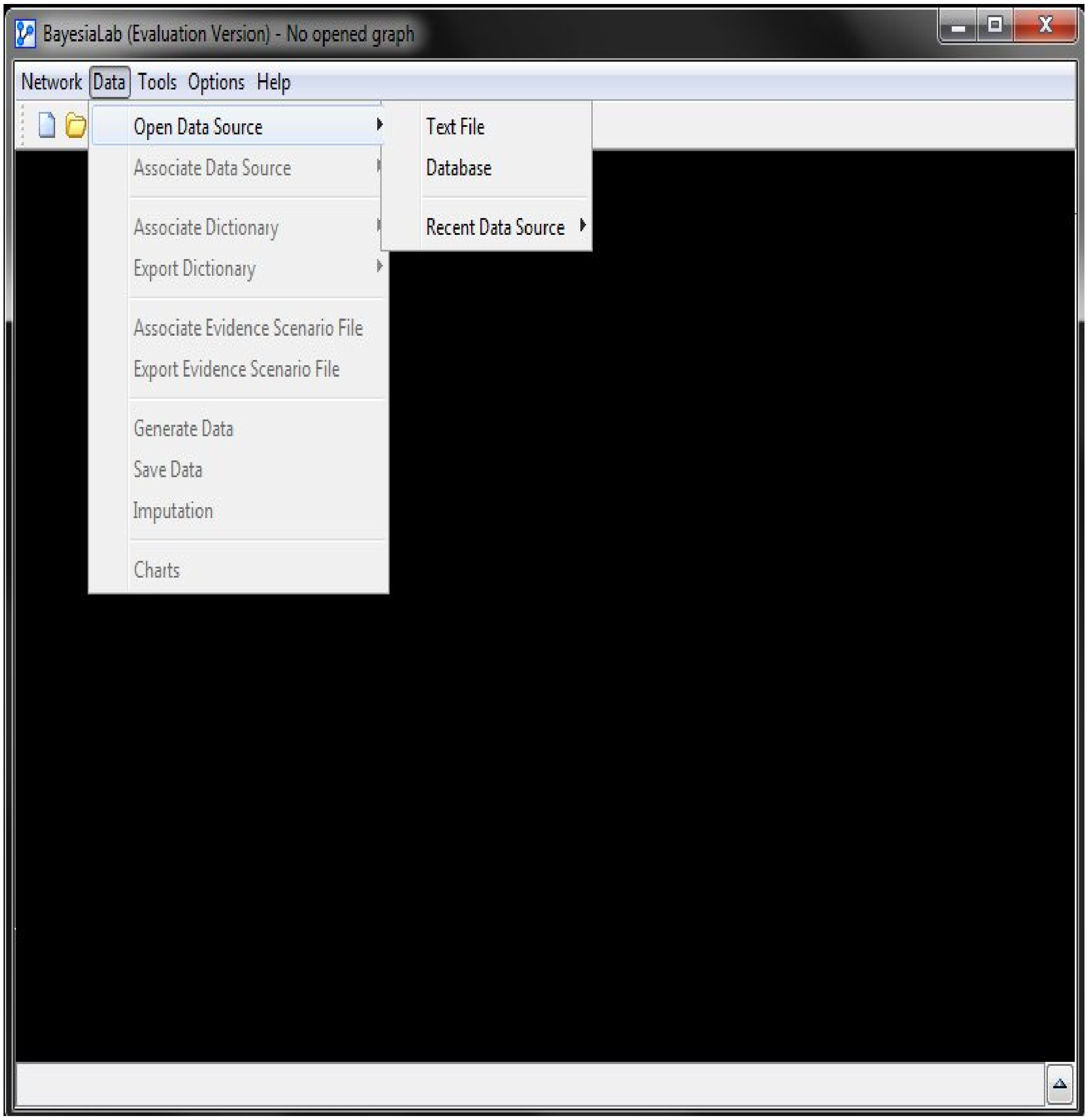


Fig 5.4.5

Select the data by choosing the text file

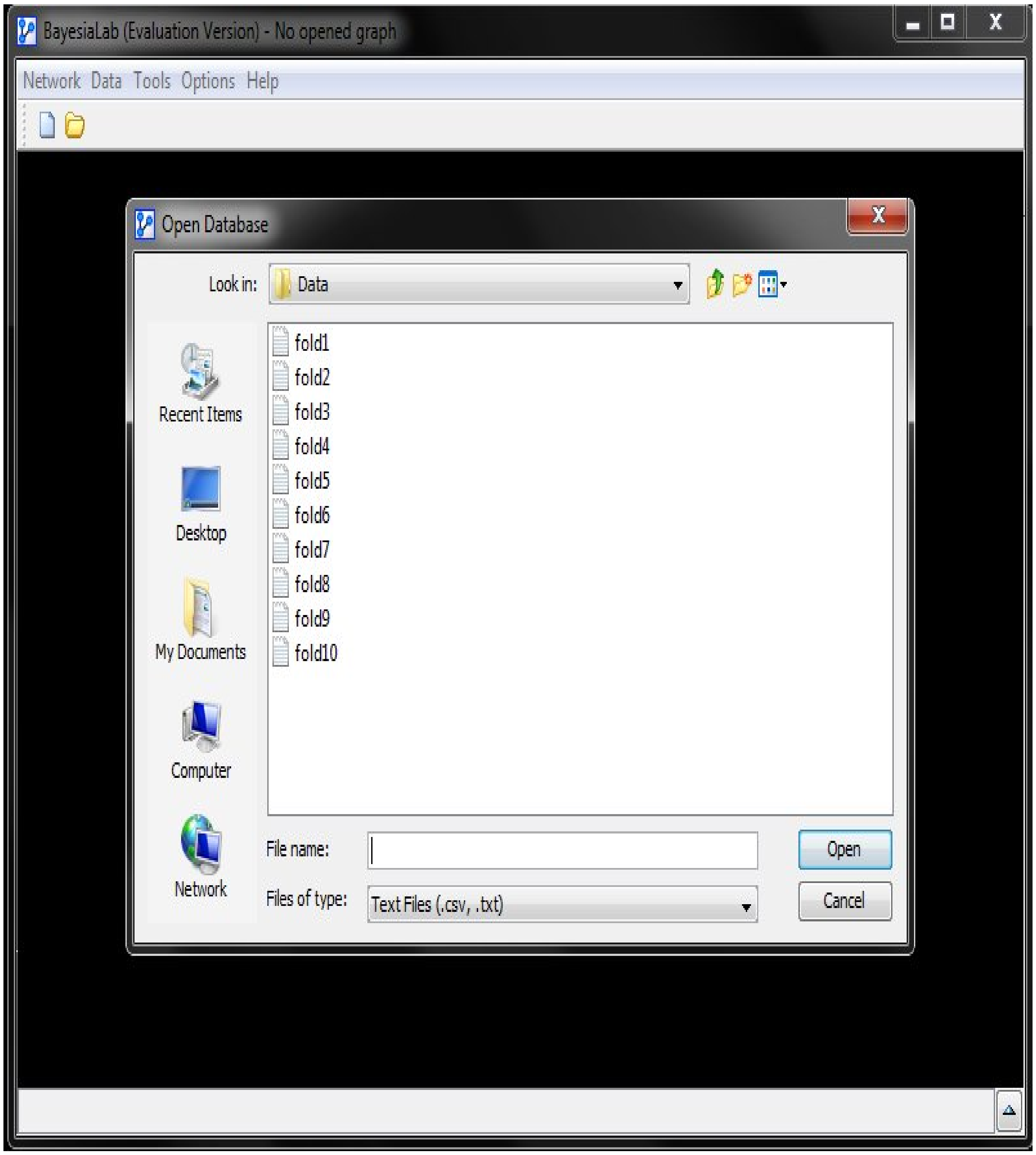
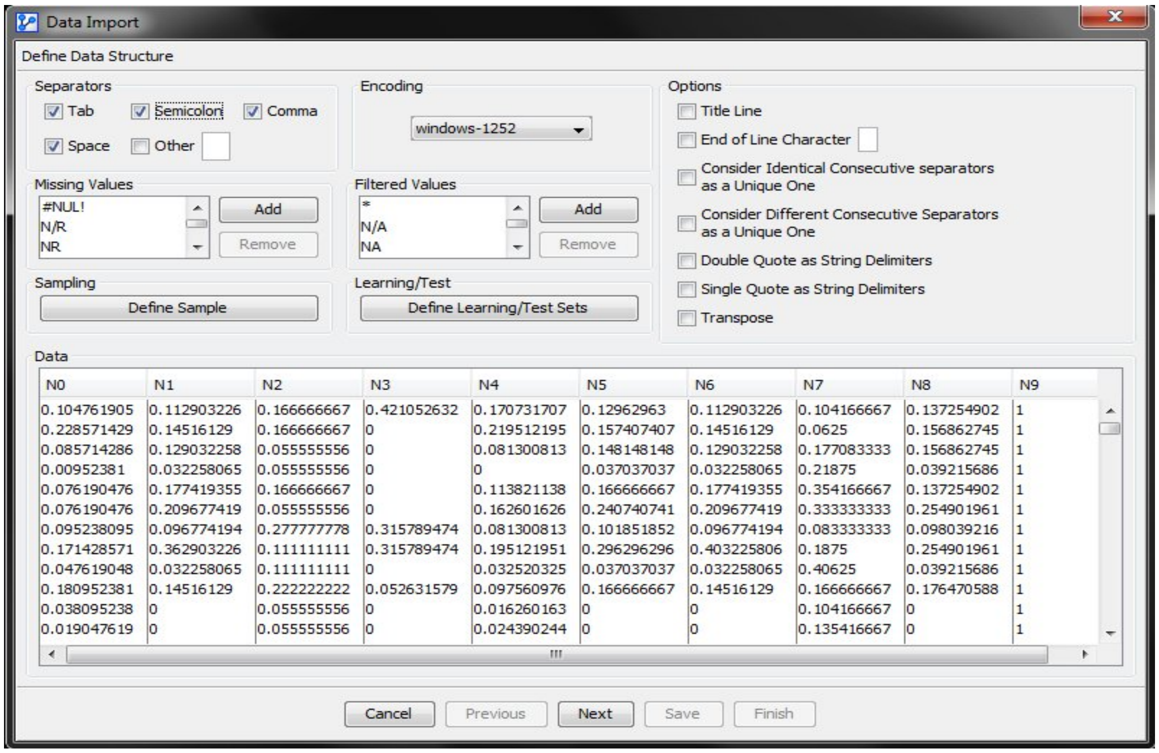


Fig 5.4.6



Data Import

Define Data Structure

Separators: [✓ Tab] [✓ Semicolon] [✓ Comma] [✓ Space] Other

Encoding: windows-1252

Options: Title Line; End of Line Character; Consider Identical Consecutive separators as a Unique One; Consider Different Consecutive Separators as a Unique One; Double Quote as String Delimiters; Single Quote as String Delimiters; Transpose

Missing Values: #NUL! N/R NR

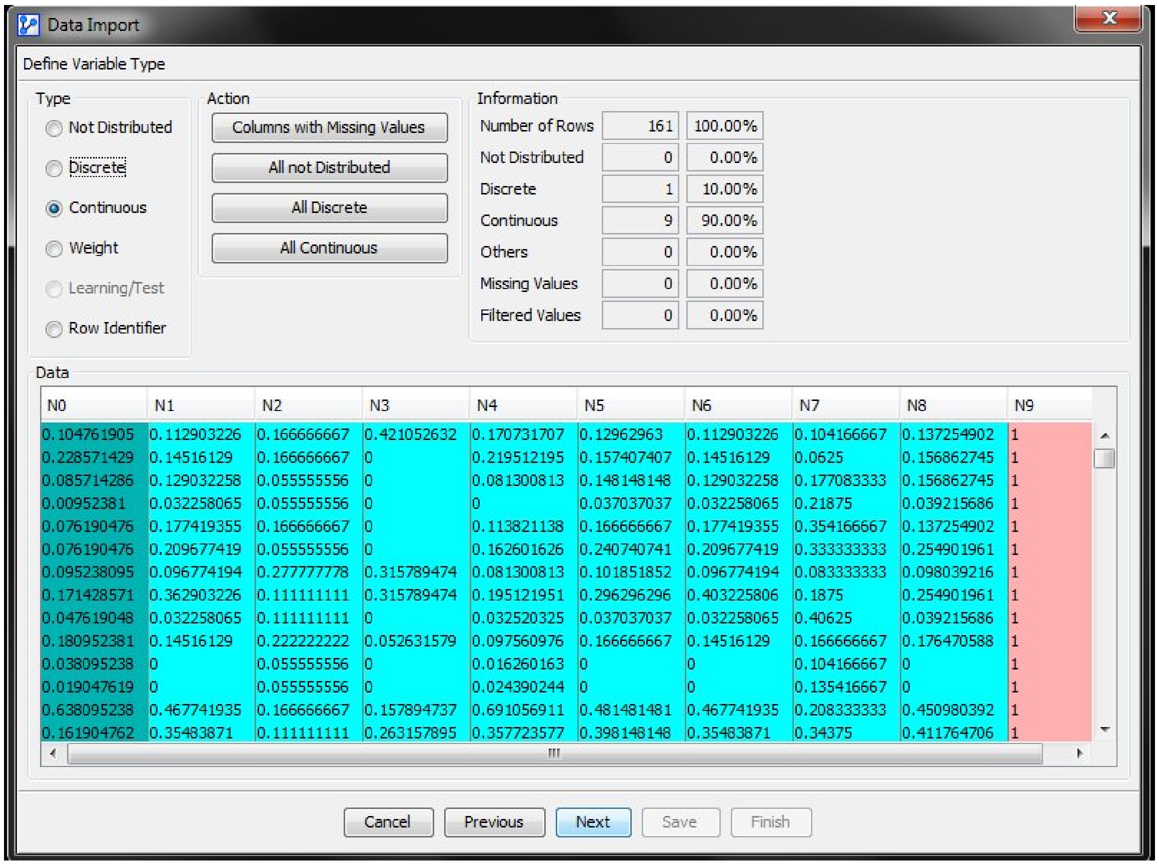
Filtered Values: * N/A NA

Sampling: Define Sample

Learning/Test: Define Learning/Test Sets

Data

N0	N1	N2	N3	N4	N5	N6	N7	N8	N9
0.104761905	0.112903226	0.166666667	0.421052632	0.170731707	0.12962963	0.112903226	0.104166667	0.137254902	1
0.228571429	0.14516129	0.166666667	0	0.219512195	0.157407407	0.14516129	0.0625	0.156862745	1
0.085714286	0.129032258	0.055555556	0	0.081300813	0.148148148	0.129032258	0.177083333	0.156862745	1
0.00952381	0.032258065	0.055555556	0	0	0.037037037	0.032258065	0.21875	0.039215686	1
0.076190476	0.177419355	0.166666667	0	0.113821138	0.166666667	0.177419355	0.354166667	0.137254902	1
0.076190476	0.209677419	0.055555556	0	0.162601626	0.240740741	0.209677419	0.333333333	0.254901961	1
0.095238095	0.096774194	0.277777778	0.315789474	0.081300813	0.101851852	0.096774194	0.083333333	0.098039216	1
0.171428571	0.362903226	0.111111111	0.315789474	0.195121951	0.296296296	0.403225806	0.1875	0.254901961	1
0.047619048	0.032258065	0.111111111	0	0.032520325	0.037037037	0.032258065	0.40625	0.039215686	1
0.180952381	0.14516129	0.222222222	0.052631579	0.097560976	0.166666667	0.14516129	0.166666667	0.176470588	1
0.038095238	0	0.055555556	0	0.016260163	0	0	0.104166667	0	1
0.019047619	0	0.055555556	0	0.024390244	0	0	0.135416667	0	1



Data Import

Define Variable Type

Type: Not Distributed; Discrete; [● Continuous]; Weight; Learning/Test; Row Identifier

Action: Columns with Missing Values; All not Distributed; All Discrete; All Continuous

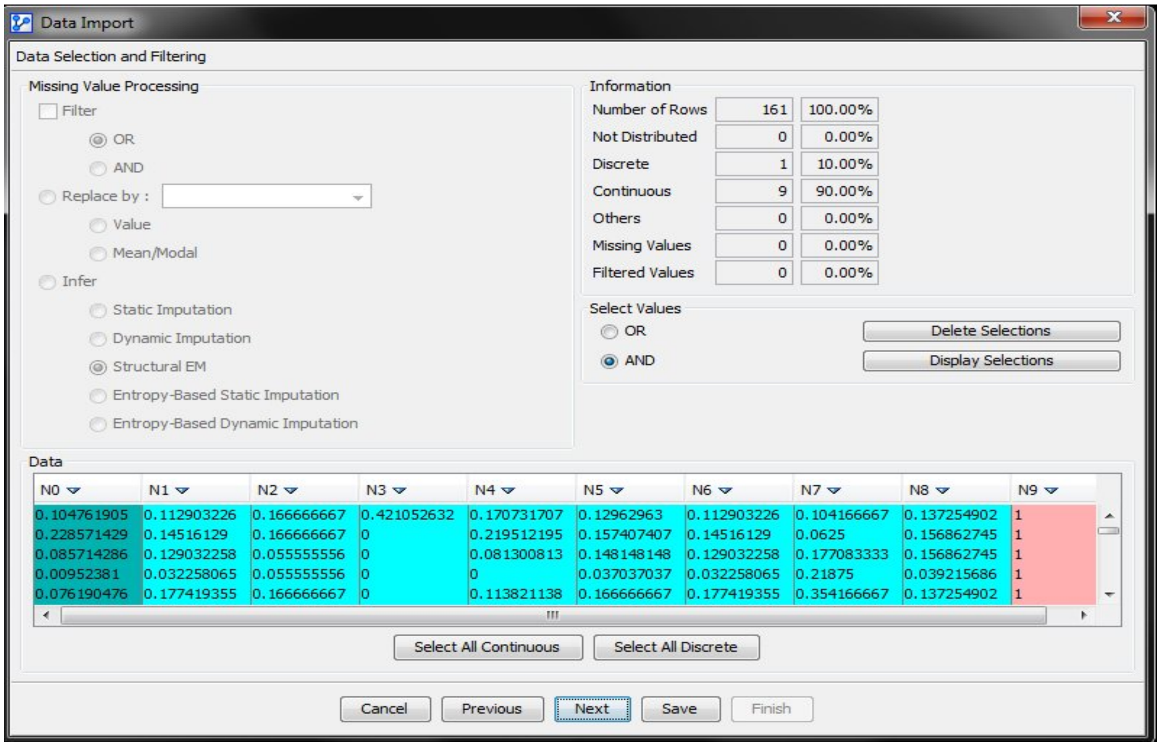
Information

Number of Rows	161	100.00%
Not Distributed	0	0.00%
Discrete	1	10.00%
Continuous	9	90.00%
Others	0	0.00%
Missing Values	0	0.00%
Filtered Values	0	0.00%

Data

N0	N1	N2	N3	N4	N5	N6	N7	N8	N9
0.104761905	0.112903226	0.166666667	0.421052632	0.170731707	0.12962963	0.112903226	0.104166667	0.137254902	1
0.228571429	0.14516129	0.166666667	0	0.219512195	0.157407407	0.14516129	0.0625	0.156862745	1
0.085714286	0.129032258	0.055555556	0	0.081300813	0.148148148	0.129032258	0.177083333	0.156862745	1
0.00952381	0.032258065	0.055555556	0	0	0.037037037	0.032258065	0.21875	0.039215686	1
0.076190476	0.177419355	0.166666667	0	0.113821138	0.166666667	0.177419355	0.354166667	0.137254902	1
0.076190476	0.209677419	0.055555556	0	0.162601626	0.240740741	0.209677419	0.333333333	0.254901961	1
0.095238095	0.096774194	0.277777778	0.315789474	0.081300813	0.101851852	0.096774194	0.083333333	0.098039216	1
0.171428571	0.362903226	0.111111111	0.315789474	0.195121951	0.296296296	0.403225806	0.1875	0.254901961	1
0.047619048	0.032258065	0.111111111	0	0.032520325	0.037037037	0.032258065	0.40625	0.039215686	1
0.180952381	0.14516129	0.222222222	0.052631579	0.097560976	0.166666667	0.14516129	0.166666667	0.176470588	1
0.038095238	0	0.055555556	0	0.016260163	0	0	0.104166667	0	1
0.019047619	0	0.055555556	0	0.024390244	0	0	0.135416667	0	1
0.638095238	0.467741935	0.166666667	0.157894737	0.691056911	0.481481481	0.467741935	0.208333333	0.450980392	1
0.161904762	0.35483871	0.111111111	0.263157895	0.357723577	0.398148148	0.35483871	0.34375	0.411764706	1

Fig 5.4.7



Data Import

Data Selection and Filtering

Missing Value Processing

- [ ] Filter
  - [✓ OR]
  - AND
- Replace by :
  - Value
  - Mean/Modal
- Infer
  - Static Imputation
  - Dynamic Imputation
  - [✓ Structural EM]
  - Entropy-Based Static Imputation
  - Entropy-Based Dynamic Imputation

Information

Number of Rows	161	100.00%
Not Distributed	0	0.00%
Discrete	1	10.00%
Continuous	9	90.00%
Others	0	0.00%
Missing Values	0	0.00%
Filtered Values	0	0.00%

Select Values: OR [✓ AND]

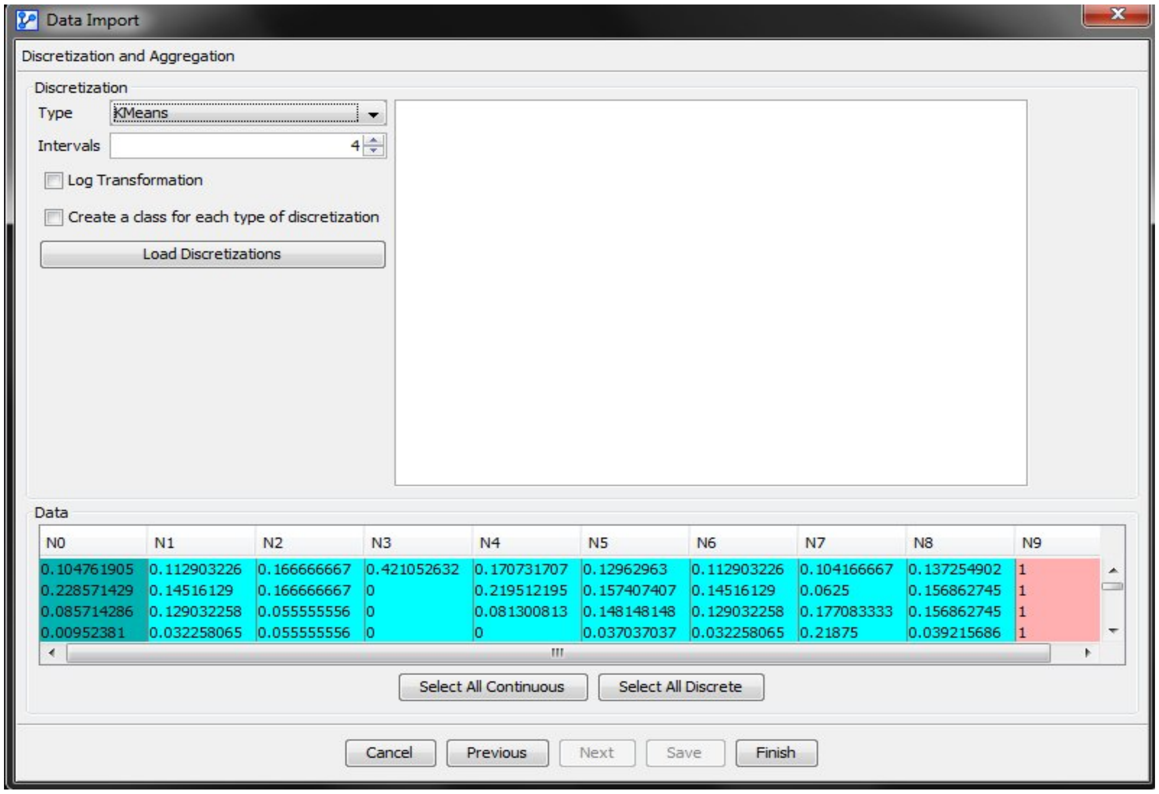
Delete Selections · Display Selections

Data

N0	N1	N2	N3	N4	N5	N6	N7	N8	N9
0.104761905	0.112903226	0.166666667	0.421052632	0.170731707	0.12962963	0.112903226	0.104166667	0.137254902	1
0.228571429	0.14516129	0.166666667	0	0.219512195	0.157407407	0.14516129	0.0625	0.156862745	1
0.085714286	0.129032258	0.055555556	0	0.081300813	0.148148148	0.129032258	0.177083333	0.156862745	1
0.00952381	0.032258065	0.055555556	0	0	0.037037037	0.032258065	0.21875	0.039215686	1
0.076190476	0.177419355	0.166666667	0	0.113821138	0.166666667	0.177419355	0.354166667	0.137254902	1

Select All Continuous · Select All Discrete

Cancel · Previous · Next · Save · Finish



Data Import

Discretization and Aggregation

Discretization

Type: KMeans

Intervals: 4

- [ ] Log Transformation
- [ ] Create a class for each type of discretization

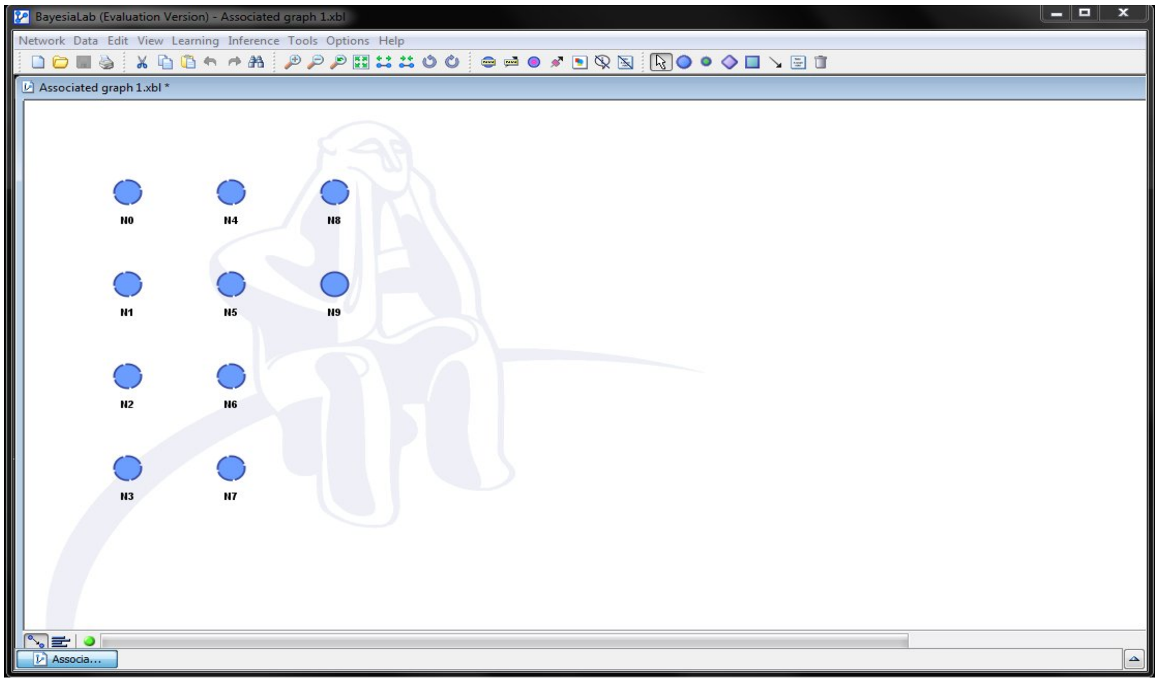
Load Discretizations

Data

N0	N1	N2	N3	N4	N5	N6	N7	N8	N9
0.104761905	0.112903226	0.166666667	0.421052632	0.170731707	0.12962963	0.112903226	0.104166667	0.137254902	1
0.228571429	0.14516129	0.166666667	0	0.219512195	0.157407407	0.14516129	0.0625	0.156862745	1
0.085714286	0.129032258	0.055555556	0	0.081300813	0.148148148	0.129032258	0.177083333	0.156862745	1
0.00952381	0.032258065	0.055555556	0	0	0.037037037	0.032258065	0.21875	0.039215686	1

Select All Continuous · Select All Discrete

Cancel · Previous · Next · Save · Finish


BayesiaLab (Evaluation Version) - Associated graph 1.xbl
Network Data Edit View Learning Inference Tools Options Help
Associated graph 1.xbl *
N0
N4
N8
N1
N5
N9
N2
N6
N3
N7
Associa...

Select N9 as Target node by right clicking on it.

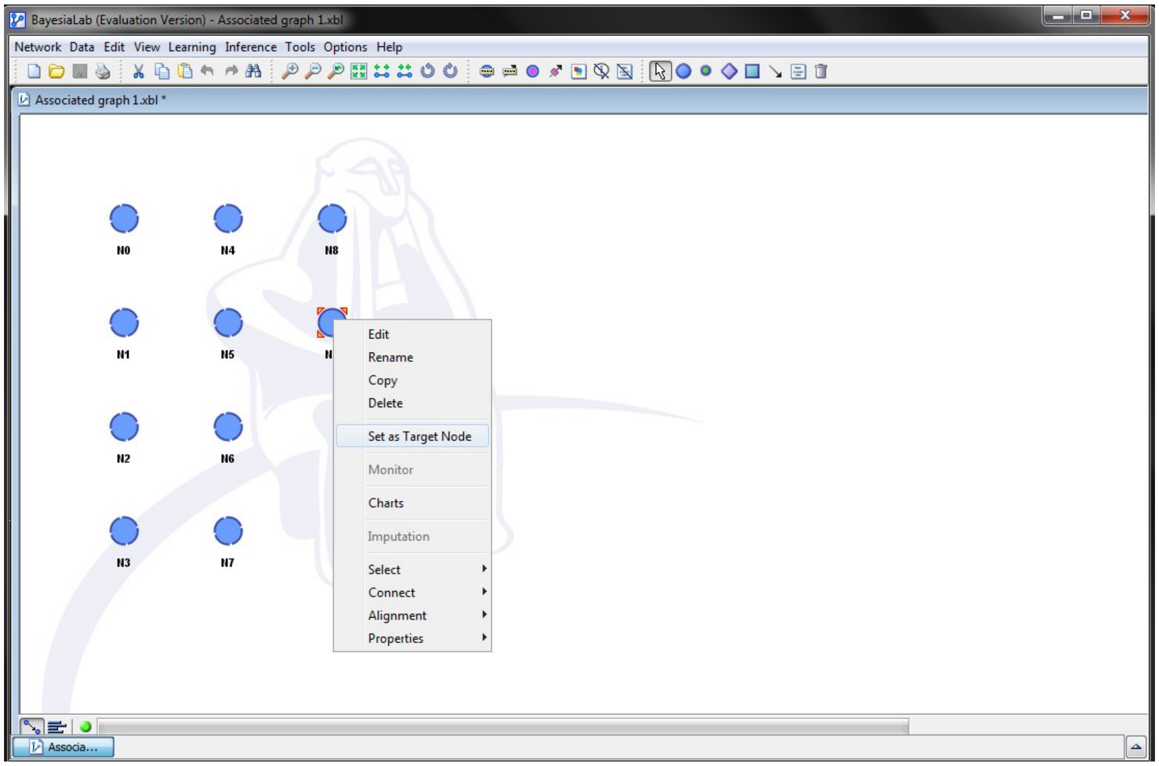


Fig 5.4.8

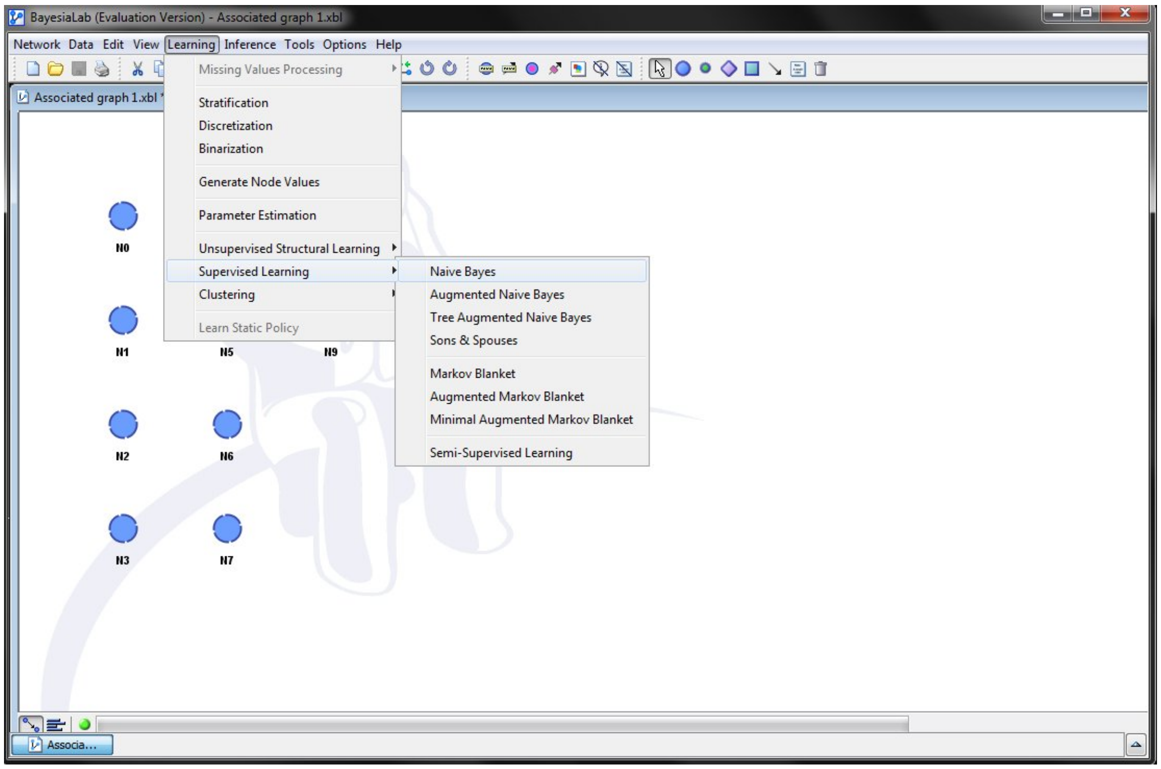
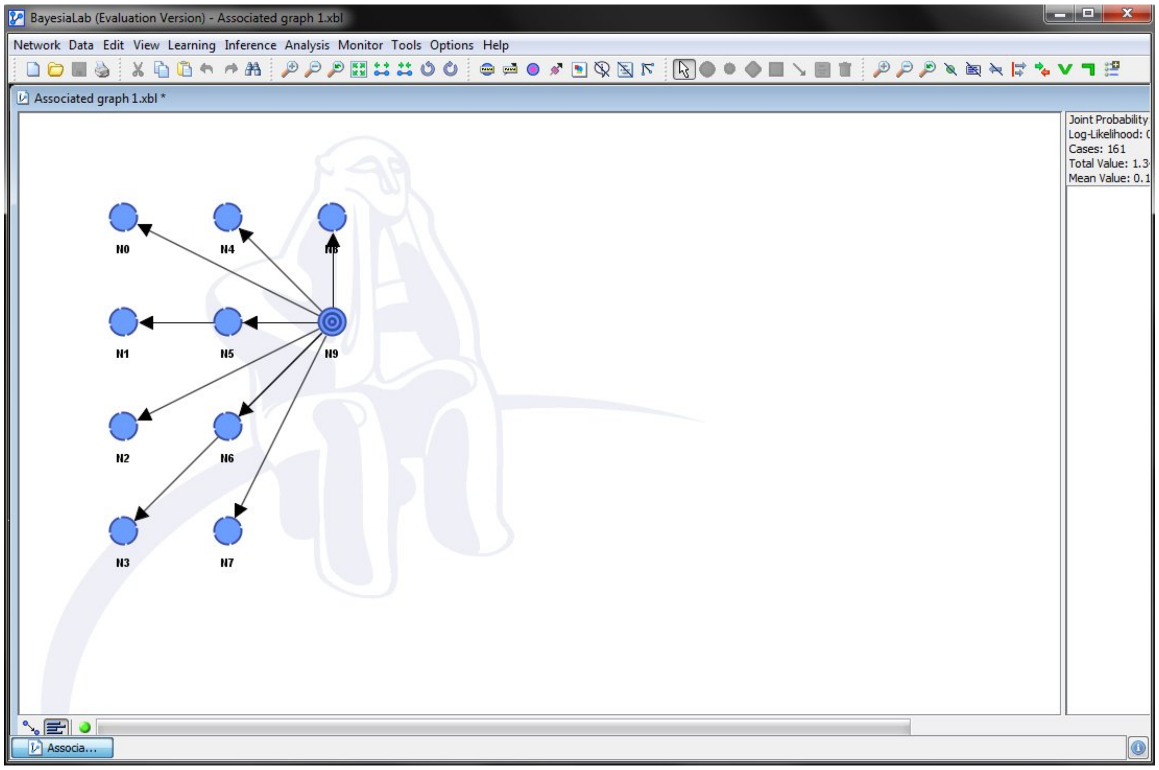
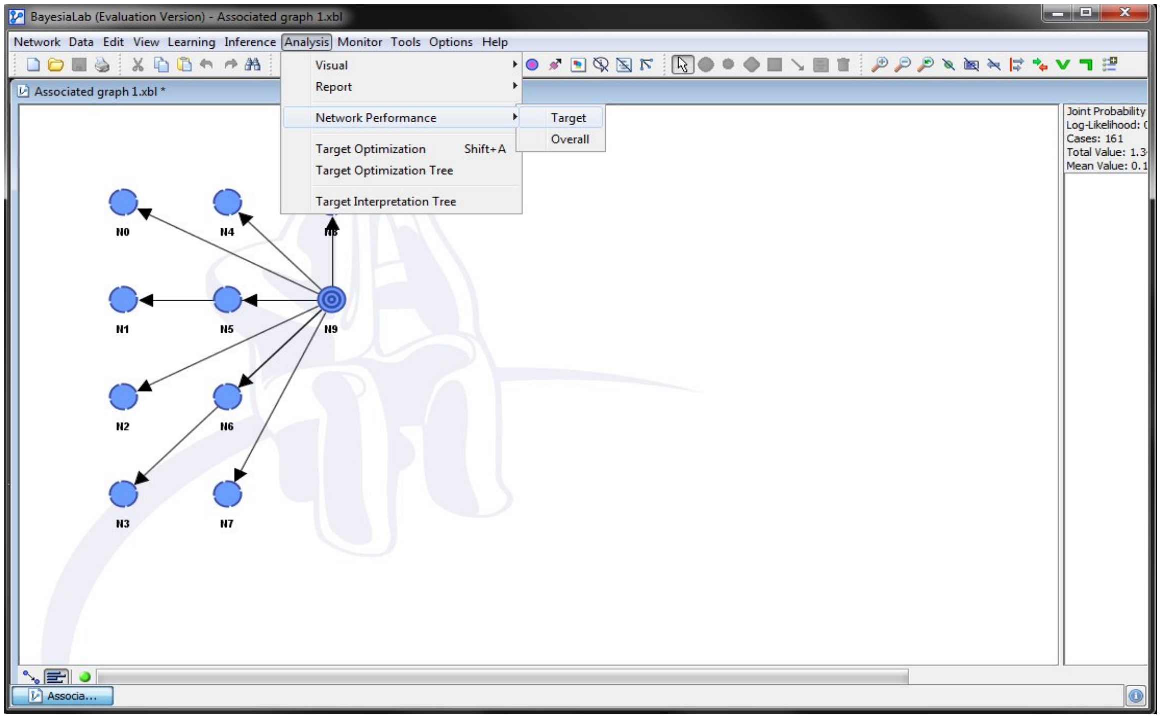

BayesiaLab (Evaluation Version) - Associated graph 1.xbl
Network Data Edit View Learning Inference Tools Options Help
Associated graph 1.xbl *
Missing Values Processing
Stratification
Discretization
Binarization
Generate Node Values
Parameter Estimation
Unsupervised Structural Learning
Supervised Learning
Clustering
Learn Static Policy
Naive Bayes
Augmented Naive Bayes
Tree Augmented Naive Bayes
Sons & Spouses
Markov Blanket
Augmented Markov Blanket
Minimal Augmented Markov Blanket
Semi-Supervised Learning
N0
N1
N5
N9
N2
N6
N3
N7
Associa...

Fig 5.4.9

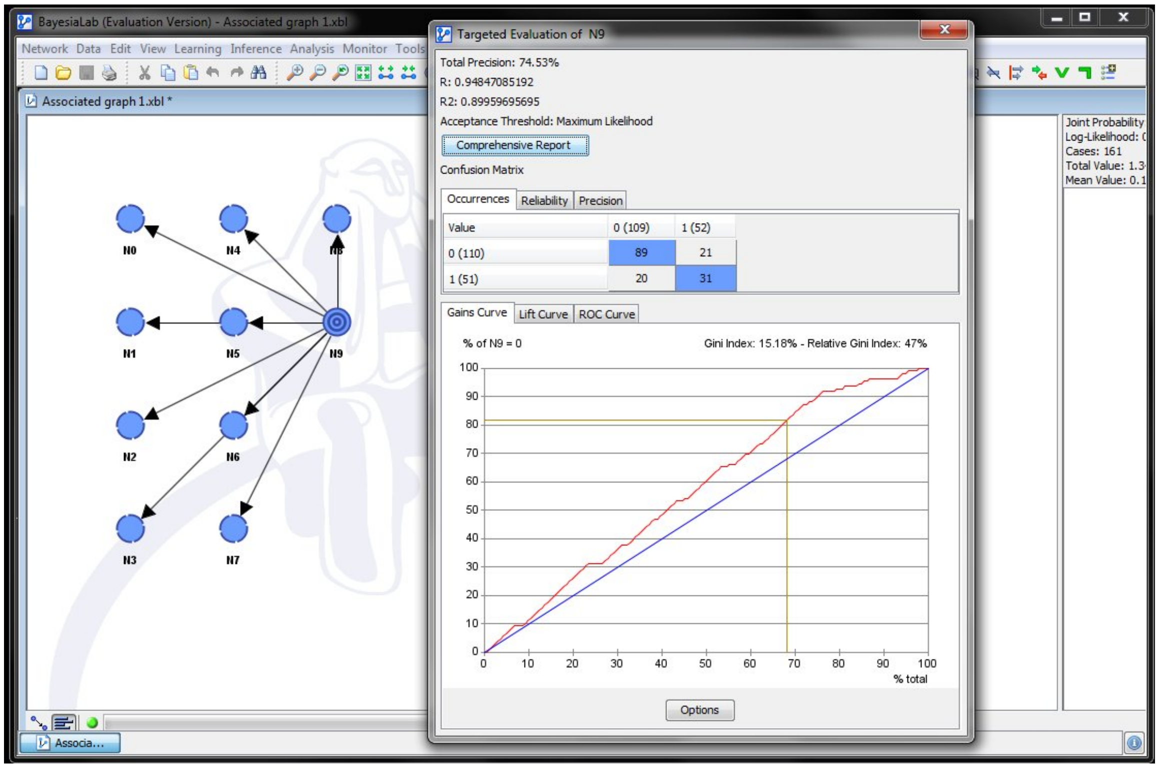

BayesiaLab (Evaluation Version) - Associated graph 1.xbl
Network Data Edit View Learning Inference Analysis Monitor Tools Options Help
Associated graph 1.xbl *
Joint Probability
Log-Likelihood: (
Cases: 161
Total Value: 1.3
Mean Value: 0.1
N0
N4
N8
N1
N5
N9
N2
N6
N3
N7
Associa...


BayesiaLab (Evaluation Version) - Associated graph 1.xbl
Network Data Edit View Learning Inference Analysis Monitor Tools Options Help
Associated graph 1.xbl *
Visual
Report
Network Performance
Target
Overall
Target Optimization Shift+A
Target Optimization Tree
Target Interpretation Tree
N0
N4
N8
N1
N5
N9
N2
N6
N3
N7
Associa...

Joint Probability
Log-Likelihood: (
Cases: 161
Total Value: 1.3
Mean Value: 0.1

Fig 5.4.10

Output



Total Precision: 74.53%
R: 0.94847085192
R2: 0.89959695695
Acceptance Threshold: Maximum Likelihood

Confusion Matrix

Value	0 (109)	1 (52)
0 (110)	89	21
1 (51)	20	31

Gini Index: 15.18% - Relative Gini Index: 47%

Joint Probability
Log-Likelihood: 0
Cases: 161
Total Value: 1.3
Mean Value: 0.1

Fig 5.4.11

CHAPTER 6

RESULTS AND DISCUSSIONS

We employed BBN available in BayesiaLab tool for text files. We collected software defect datasets. The datasets are comprised of several software modules with their attributes. The average results obtained in our experiments as follows:

Table 1 – Sensitivity : 79.6% , Specificity : 58.1% , Accuracy : 72.8%

Table 2 – Sensitivity : 77% , Specificity : 59.3% , Accuracy : 72.3%

Table 3 – Sensitivity : 75.5% , Specificity : 46.8% , Accuracy : 71.2%

Table 6.1 - Naïve Bayes Results

	Sensitivity	Specificity	Accuracy
FOLD 1	80	58	73
FOLD 2	80	58	73
FOLD 3	81	59	73
FOLD 4	80	57	72
FOLD 5	80	55	78
FOLD 6	77	57	70
FOLD 7	79	60	72
FOLD 8	80	60	71
FOLD 9	80	57	74
FOLD 10	79	60	72
Average	79.6	58.1	72.8

Table 6.2 – Markov Blanket Results

	Sensitivity	Specificity	Accuracy
FOLD 1	78	57	72
FOLD 2	78	57	72
FOLD 3	78	57	72
FOLD 4	74	70	75
FOLD 5	78	57	72
FOLD 6	74	68	73
FOLD 7	78	57	72
FOLD 8	78	57	72
FOLD 9	78	57	72
FOLD 10	76	56	71
Average	77	59.3	72.3

<u>Table 6.3</u> – Augmented Markov Blanket Results

	Sensitivity	Specificity	Accuracy
FOLD 1	78	57	72
FOLD 2	78	56	71
FOLD 3	77	57	72
FOLD 4	68	0	68
FOLD 5	74	70	73
FOLD 6	78	57	72
FOLD 7	78	57	72
FOLD 8	78	57	72
FOLD 9	78	57	72
FOLD 10	68	0	68
Average	75.5	46.8	71.2