Bayesian Approach to Cognitive Systems

D4.3: First improved version of the tool box

Due date of deliverable: 30-06-2007
Actual submission date: 30-06-2007

Start date of project: 01.01.2006
Duration: 50 month

Organisation name of lead contractor for this deliverable: Probayes
Revision [1,]

<table>
<thead>
<tr>
<th>Dissemination Level</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>PU</td>
<td>Public</td>
</tr>
<tr>
<td>PP</td>
<td>Restricted to other programme participants (including the Commission Services)</td>
</tr>
<tr>
<td>RE</td>
<td>Restricted to a group specified by the consortium (including the Commission Services)</td>
</tr>
<tr>
<td>CO</td>
<td>Confidential, only for members of the consortium (including the Commission Services)</td>
</tr>
</tbody>
</table>
Contents page

1 INTRODUCTION ........................................................................................................................................ 4

2 THE APPLICATION PROGRAM INTERFACE .................................................................................... 6
  2.1 SAMPLE SPACE MODULE.................................................................................................................... 6
  2.2 PROBABILITY DISTRIBUTIONS Module ............................................................................................ 6
  2.3 EXPERIMENTAL-DATA-LEARNING Module .................................................................................... 8
    2.3.1 Data source descriptors ............................................................................................................. 8
    2.3.2 Maximum a posteriori learning ............................................................................................. 8
    2.3.3 Maximum likelihood estimate ............................................................................................. 9
    2.3.4 Expectation-Maximization algorithm ..................................................................................... 9
    2.3.5 Model learner ...................................................................................................................... 9
  2.4 SERIALIZATION Module .................................................................................................................. 9

3 THE BAYESIAN INFERENCE ENGINE ............................................................................................... 11

4 PROFILING TOOLS .......................................................................................................................... 12
Summary

This document reports the main improvements contained in the new version of ProBT©. The choice of the improvements roadmap was leaded by the main requirements of BACS partners. ProBT© users will find a set of new functionalities that facilitates the creation of Bayesian models and their reusability. This set includes new learning capabilities, a new optimization technique, a serialization module, the integration of a C scripting language, and a profiling tool called ProBT© XL.

This work has been possible thanks to the support of the European BACS project.

The improved new version of the ProBT© toolbox is available for all BACS partners at:

http://bacs.probayes.com
1 Introduction

For the new version of ProBT©, Probayes improved the existing modules and integrated new ones. Figure 1 shows a global view of the current version of the ProBT© library. The main improvements concern: (i) the inference and learning algorithms, (ii) the integration of a serialization module and (ii) the creation of an external profiling tool.

In the sample space module, a new type of variables is available: the interval variables. This kind of variable has proved to be very useful in a variety of applications.

In the experimental data learning module, a set of conjugate laws was introduced. These conjugate laws allow effortlessly Bayesian learning. The corresponding C++ classes are now available on the ProBT© Application Program Interface (API) and the required computations are embedded in the ProBT Bayesian Inference Engine (BIE).

In the integration and optimization module, a new optimization technique was developed. The optimization technique allows faster probability calculus computation.

Another major improvement is the serialization of ProBT© objects. In fact, a big number of applications require saving or transmitting ProBT© objects, for example onto a file, a memory buffer, or across the network. The goal is to reuse these objects into other programs or processors for incremental modelling or computation.

Probayes integrated a runtime C compiler in the new version of ProBT©. The embedded compiler allows the definition of user distributions by means of a C script. The main interest of this compiler is that the run time capability allows the serialization of the user-defined distributions and consequently its exchange and execution independently of the computer platform.

Within the axe of profiling tools a graphical interface was developed. This interface allows the definition and execution of Bayesian models without using C++ programming. The interface has been embedded in Excel® which calls an external ProBT© library for the execution of the programs. The interface is intended for fast Bayesian modelling and mainly conceived to be used by those programmers who are not experts in C++.

Figure 2 depicts a graphical representation of the new ProBT© class hierarchy, new API components are highlighted. Evidently, for each of these classes, the corresponding implementation on the BIE was also done.
Figure 2: Global view of the current ProBT© API class hierarchy. Highlighted classes corresponds to new elements with respect to the previous ProBT© version.
2 The application Program Interface

2.1 Sample Space Module

A new type of variables has been included in ProBT®: the variables of type interval. In the upcoming version of ProBT®, these new variables will integrate specialized probability calculus (e.g. integrals computation) that will improve the current time and quality computation of the programs containing them. The class plIntervalType implements this new type.

The object class plVariableIndexer was introduced on the new version of ProBT. A variable indexer allows the user to retrieve the cardinality of the \( n \)th symbol from an object of type variables conjunction (plVariablesConjunction). Objects of type plVariableIndexer are very useful, especially for programmers that implement functionalities with dynamic variables conjunction.

2.2 Probability Distributions Module

In order to construct extensive Bayesian models in ProBT© it is necessary to allow the construction of user-defined distributions. The new improved version of ProBT© proposes a new way of defining such distributions.

In the previous version of the ProBT© API, a user-defined distribution was defined either as a C++ function or as a C++ object class method. This means that the ProBT© C++ application must be linked with the user files containing the functions or classes. If the user function or method changed, the application should be recompiled. Otherwise stated, the definition or redefinition of user-defined distributions was not possible at runtime.

The new improved version of ProBT© interacts with a run time C compiler. From now on, a ProBT© Bayesian program can use a script in C language to define user-defined distributions. This functionality is very useful in Bayesian modelling, especially when the model is constructed and executed from a graphical human machine interface for instance Excel© (see Section 4) or when the Bayesian program should be sent to another processor for execution. The scripting language allows integrating the user functions alongside the Bayesian model and by consequence its serialization (see section XXX).

User functions defined with the C script language are then accessible and executable by any program compiled with the ProBT© library. Furthermore, the user-defined distributions can be modified and recompiled without necessity of stopping the application. The ProBT© API classes plExternalFunctionFromC and plExternalProbFunctionFromC allow to implement such functionalities.

Figures 3 and 4 show two analogous programs. The first one shows how to create an object representing the conditional distribution \( P(\text{Points} \mid \text{Die}_1, \text{Die}_2 \ldots \text{Die}_n) \) with a C++ user function and the second one shows how to create an equivalent object by using the C scripting language.
void add_dice(plValues &dice_addition, const plValues &die) {
    unsigned int n_dice;   // Number of dice
    unsigned int sum=0;    // sum of points
    unsigned int i;
    n_dice = die.size();   // Get the number of dice
    for(i=0;i<n_dice;i++)  // Compute the sum of points
        sum = sum+die[i];
    dice_addition[0] = sum;     // Set the output values
}

int main() {
    // Create the external function
    plExternalFunction sum(points,dice,add_dice);
    // P(points=p | Die0=die0 Die1=die1 ... Die(n-1)=die(n-1)) =
    // 1 if points = add_dice(die0,die1,...,die(n-1)) else 0
    plFunctionalDirac p_addition(points,dice,sum);
    ...
}

Figure 3: Code allowing to construct a distribution P (Points | Die₁, Die₂ ... Dieₙ) with the user-defined C++ function “add_dice”. Once the program in execution it is no more possible to change the code of “add_dice”. In addition, all programs that would like to use this function must be linked with the add_dice function.

string my_code =
    "void add_dice(double* output, unsigned int output_size, \n    double* input,  unsigned int input_size)\n    {\n        double* dice_addition = output;\n        double* die = input;\n
        unsigned int n_dice;   /* Number of dice */
        double sum=0.0;    /* sum of points */
        unsigned int i;\n
        n_dice = input_size;   /* Get the number of dice */

        for(i=0;i<n_dice;i++)  /* Compute the sum of points */
            sum = sum+die[i];\n
        dice_addition[0] = sum; /* Set the output values */\n    }\n";  

int main() {
    // Create the external function
    plExternalFunctionFromC sum(points,dice,my_code,"add_dice");
    // P(points=p | Die0=die0 Die1=die1 ... Die(n-1)=die(n-1)) =
    // 1 if points = add_dice(die0,die1,...,die(n-1)) else 0
    plFunctionalDirac p_addition(points,dice,sum);
    ...
}

Figure 4: An analogous program to that of Figure 3. The string variable “my_code” contains the equivalent code to that of the “add_dice” function on Figure 1. In this case, the code is “captured” into a string and it is executed using the C scripting language. The code in the string variable could be retrieved from another source, for example, from a human machine interface, a network link or a file. In this case, the code can be changed on run time.
2.3 Experimental-Data-Learning Module

Learning from data is a central issue in Bayesian modelling, that is why, we gave particular interest to this axe of development in the new version of ProBT©. In the following paragraphs, we describe the new ProBT© learning capabilities that include five different axes:

a. Data source descriptors
b. Maximum a posteriori learning
c. Maximum likelihood learning
d. Expectation-maximization algorithm
e. Model learner

2.3.1 Data source descriptors

A data source descriptor allows retrieving data from different sources. Data source descriptors are essential for facilitating the use and the reusability of the learning algorithms. In the new improved version of ProBT©, the data source descriptors were integrated in the API by the abstract class “plDataDescriptor”.

To allow this abstraction in ProBT©, the learning functions and classes accept instances of such data descriptors as input. Examples of these functions are:

- `plLearnObject::learn_using_data_descriptor` for online learning,
- `plELearner::run` for batch learning using the EM algorithm.

Besides the abstract class “plDataDescriptor”, the new ProBT© API provides some specialized classes for:

- Data matrices (`plMatrixDataDescriptor` class),
- CSV files (`plCSVFileDataDescriptor` class),
- MySQL databases (`plMysqlDataDescriptor` class).

Of course, this architecture allows implementing new data descriptors for other sources of data if needed.

2.3.2 Maximum a posteriori learning

A classical problem on Bayesian inference consists of estimating the parameters $\theta$ of a distribution given the observed data $D$. To solve this problem the joint distribution of $\theta$ and $D$ is required:

$$ P(\theta, D) = P(\theta) P(D | \theta) $$

The distribution $P(\theta)$ is said to be the prior distribution of $\theta$ and $P(D | \theta)$ is said to be the sampling (likelihood) distribution. By using the Bayes’ theorem it is possible to obtain the following expression:

$$ P(\theta | D) \propto P(\theta) P(D | \theta) $$

The distribution $P(\theta | D)$ is the posterior density function of $\theta$ and it estimates the unknown parameter $\theta$. Conjugacy is a particular property of this kind of inference. Conjugacy exists if the posterior distribution $P(\theta | D=d)$ has the same parametric form than the prior distribution $P(\theta)$. The advantage of conjugacy is that the computation of the posterior distribution (the computation $\theta$) has a closed form solution resulting in a relaxation of the computation.

In the axis of learning algorithms the new version of ProBT© integrates two new conjugate laws: `plBayesLearnProbTable` and `plBayesLearnGamma`. 
2.3.3 Maximum likelihood estimate

The maximum likelihood estimate (ML) allows parameters estimation by finding the distribution (from a family of distributions) that maximizes the probability of observing the given data. Note that, in this case no assumption about the distribution of the parameters \( \theta \) is done. For the new version of ProBT©, we have added an additional ML learning class the \texttt{plLearnBellShape}. This class allows learning the parameters of a normal-like distribution on an integer variable.

2.3.4 Expectation-Maximization algorithm

The Expectation-Maximization (EM) algorithm is an iterative algorithm that allows Bayesian model parameters estimation. The EM algorithm executes two main steps: (i) compute the expectation of the likelihood; (ii) replace the initial estimate of the parameters with the value that maximizes the likelihood. These two steps are iterated until a convergence condition is reached, for example, once the estimated parameters change less than a given value \( \epsilon \). EM algorithm is very useful in many classical problems for instance mixture models. EM algorithm is particularly efficient when the estimated model can be expressed as a set of distributions where the posterior density functions have closed-form solutions.

The new version of ProBT© includes an object class that implements a generic EM learning algorithm. An object of this class is constructed with four parameters: (i) the model variables, (ii) the initial distributions, (iii) a vector of incremental learning objects with the corresponding parametrical and corresponding priors, and (iv) a pointer to the data descriptor to be used for learning.

2.3.5 Model learner

In Bayesian learning, we can be confronted to learn the parameters of a full joint distribution of a model from complete or incomplete data. ProBT© includes now two classes allowing this kind of learning:

- \texttt{plModelParamsCompleteDataLearner} for complete data
- \texttt{plModelParamsIncompleteDataLearner} for incomplete data

2.4 Serialization Module

A new functionality has been integrated in ProBT©: the serialization of the ProBT© objects. Serialization is intended to saving or transmitting the API objects, for example, onto a file, a memory buffer or across a network connection link. Serialization allows the reconstruction of ProBT© objects that have been constructed in the past by the execution of a third program.

The main goal is to reuse these objects into other programs or processors for incremental modelling and computation, or for their transmission as a “ready-to-use” program. For example in robotics applications, we can imagine that the robot is allowed to execute a learning phase in two (or more) disjoint training time intervals. Once the first learning phase is executed, the robot can save its knowledge as a ProBT© object (e.g. distribution). This is done without storing the raw data that generated the learned object. For the second training time interval, the robot will be capable of using the previous learned object by loading it in its memory. The new raw data will be integrated in the learned object as if the two training intervals were a single one.

Another application example is the parallelization of Bayesian programs. Indeed, an important number of applications require a large number of computations. Once a Bayesian model is constructed the serialization allows transferring the model to different processors so that each of the processors executes part of the probability calculus resulting in the reduction of the computation time.
Serialization can also be used for reusability of large Bayesian models. In fact, when inferring a distribution, part of the computation time is expended in the symbolic simplification phase. Once the question simplified, it can be saved and then reused avoiding the re-execution of the simplification.

The new version of ProBT© allows the serialization of its objects in XML format. Figure 5 shows an example of the resulting XML file when serializing the objects of the program shown on Figure 4.

```xml
<xml_document>
  <xml_document_name>
    <xml_function_class_id="26" class_name="kplExternalFunctionFromC" tracking_level="1" version="0" object_id="21">
      <xml_name>add_dice</xml_name>
      <xml_source_code>
        void add_dice(double* output, unsigned int output_size, double* input, unsigned int input_size) { double* dice_addition = output; double* dice_input = input; unsigned int n_dice; /* Number of dice */ double sum=0.0; /* sum of points */ unsigned int i; a_dice = input_size; /* Get the number of dice */
        for(i=0;i<n_dice;i++) /* Compute the sum of points */ sum += get_die()[i]; dice_addition[i] = sum; /* Set the output values */}
      </xml_source_code>
      <xml_search_variables>
        <xml_variable_class_id="26" object_id="21">
          <xml_variable_name>add_dice</xml_variable_name>
        </xml_variable_class_id="26" object_id="21">
        <xml_variable_class_id="25" object_id="21">
          <xml_variable_name>add_dice</xml_variable_name>
        </xml_variable_class_id="25" object_id="21">
      </xml_search_variables>
      <xml_known_variables>
      </xml_known_variables>
    </xml_function_class_id="26" class_name="kplExternalFunctionFromC" tracking_level="1" version="0" object_id="21">
  </xml_document_name>
</xml_document>

Figure 5: Part of the corresponding XML file generated when serializing the program on Figure 3.
3 The Bayesian Inference Engine

The Bayesian inference engine has been improved by integrating the corresponding algorithms that executes the computations required by the new API functionalities. Mainly the inference and optimization modules were improved.

The inference module has been augmented with the new learning algorithms. The optimization module integrated a new optimization technique. The developed algorithm is a new Genetic Algorithm that does not require encoding and decoding processes as required in traditional Genetic Algorithms. Consequently, the probability calculus computations are accelerated. In addition, the algorithm allows the exploration of the search space in a continuous way.
4 Profiling Tools

The development of a profiling tool was started. A first version of a graphical interface module in Excel® has been developed. We call this interface ProBT© XL. This interface has been conceived to design Bayesian networks while taking advantage of the Excel environment and the Bayesian programming approach.

ProBT© XL provides state of the art inference and learning mechanisms allowing building complex Bayesian Models. Using ProBT© XL, it is possible to perform exact or approximate inference, learn distributions from data, define object level Bayesian networks, perform dynamic inference and include user-defined functions. ProBT© XL is well suited to test Bayesian models using the Excel© facilities to handle the data and evaluate the results. Figure 6 shows a snapshot of the ProBT© XL interface.

![ProBT© XL interface](image)

**Figure 6:** The ProBT© XL interface allows the construction of Bayesian models; it takes advantage of the Excel© facilities to handle data and evaluate results.

ProBT© XL allows the translation of a Bayesian Network in its equivalent Bayesian program and vice versa. ProBT© XL models can be exported to applications linked with the ProBT© library. This feature allows using the ProBT© XL designed models outside the Excel environment. For example, the user could design different Bayesian models with the ProBT© XL interface and execute them in a more powerful computer (see Figure 7). The execution is possible thanks to the serialization module that transforms the model constructed with ProBT© XL into its equivalent in XML file. The XML file is transmitted to a second computer in order to execute the required computations.
Figure 7: A Bayesian model is constructed with the ProB® XL graphical interface. Its representation in XML is transmitted to a powerful machine for computation. The model can be modified and executed as many times as desired without need of any recompilation.