

# POULIN–HUGIN

## Patterns and Predictions

Version 1.6

# P-H Patterns and Predictions Manual

---

**Copyright 1990–2007 by Hugin Expert A/S and Poulin Holdings LLC.  
All Rights Reserved.**

This manual was prepared using the  $\text{\LaTeX}$  Document Preparation System and the  $\text{\PDF\TeX}$  typesetting software.

Set in 11 point Bitstream Charter, Bitstream Courier, and AMS Euler.

Version 1.6, September 2007.

UNIX is a trademark of The Open Group

POSIX is a trademark of IEEE

Sun, Solaris, Sun Enterprise, and Java are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and in other countries.

Microsoft, Visual C++, Windows, Windows 95, and Windows NT are trademarks or registered trademarks of Microsoft Corporation in the United States and/or other countries.

Mac OS is a registered trademark of Apple Computer, Inc.

X Window System is a trademark of X Consortium, Inc.

Bitstream and Bitstream Charter are registered trademarks of Bitstream Inc.

Hugin Expert, the Hugin Raven Logo, Hugin API, Hugin Regular, and Hugin Runtime are trademarks of Hugin Expert A/S.

POULIN HUGIN Automation Tool is a trademark of Hugin Expert A/S and Poulin Holdings LLC.

# Preface

The “P-H Patterns and Predictions manual” provides a reference manual for the P-H Patterns and Predictions package.

The present manual assumes limited familiarity with the methodology of Bayesian analysis based on Bayesian belief networks.

## Overview of the manual

**Chapter 1** gives a brief introduction to the P-H Patterns and Predictions package. It also gives some general information on the tools contained in the software package.

**Chapter 2** explains the tools for data preparation and manipulation. These tools may be used to format your data according to the HUGIN data specification format.

**Chapter 3** provides the tools for constructing and updating statistical models.

**Chapter 4** describes the tools for inference and analysis on statistical models.

**Appendix A** gives examples on how tools of the P-H Patterns and Predictions software package may be used on three different sets of data.

**Appendix B** gives examples on how tools of the P-H Patterns and Predictions software package may be used for data normalization.

**Appendix C** gives answers to frequently asked questions on the use of the P-H Patterns and Predictionstools.

## A note on the tool descriptions

The tool descriptions in this manual are presented in terms of tool prototypes, giving the names, options, and arguments of each command-line tool.

The notation “*dat2hcs*<sup>(8)</sup>” is used to refer to a command-line tool (in this case, the *dat2hcs* tool). The parenthesized, superscripted number (8) refers to the page where the command-line tool is described.

## Overview of new features in P-H Patterns and Predictions version 1.1

- The *ph -whatif* command has been split into two separate commands *ph -whatifvariable* and *ph -whatifvariables*, respectively, in order to make the distinction between the two explicit. The first command supports what-if analysis on the hypothesis variable with respect to a single variable and an evidence case whereas the second command supports what-if analysis on the hypothesis variable with respect to all observed variables in the case.
- A save-output-to-file option has been added to the following commands: *ph -inference*, *ph -voicase*, *ph -voivvariables*, *ph -whatifvariable*, and *ph -whatifvariables*. The save-output-to-file option allows the user to direct the output of the aforementioned commands to a file instead of standard output. This is useful for logging and subsequent inspection of results.
- A new option *-wpunct* has been added to the *ustruct2dat* command. The optional argument *-wpunct* specifies whether or not punctuation should be extracted from the HTML file. The default behavior is to remove punctuation.
- The P-H Patterns and Predictions manual has been revised to give more intuition on the command-line tools and the manual now includes a frequently asked questions appendix.

## Overview of new features in P-H Patterns and Predictions version 1.2

- The discretization process has been adjusted such that discretized continuous variables cover the full range from minus infinity to infinity.

- A new command, *accuracy*, for determining the classification accuracy has been added to the suite of tools.
- A new command, *csv2dat*, for merging a set of CSV formatted data files into a single HUGIN data file has been added to the suite of tools.
- A new command, *feature*, for feature selection based on statistical tests for independence has been added to the suite of tools.
- A new command, *fit2net*, for fitting a data file to a HUGIN network model has been added to the suite of tools.

### Overview of new features in P-H Patterns and Predictions version 1.3

- A new option, *-hnbm*, has been added to the *ph* command. This option supports the construction of Hierarchical Naive Bayes Models. This is useful for identifying potential hidden variables in data.
- The *accuracy* command for determining the accuracy of a classification model has been adjusted to implement n-fold cross validation.
- Two new alternative discretization processes have been added to the *ph* tool. The user may now select between uniform, supervised, and unsupervised discretization of continuous variables when using one of the *-nbm*, *-hnbm*, *-tan*, or *-update* options. Default is unsupervised discretization.

### Overview of new features in P-H Patterns and Predictions version 1.4

- To simplify the use of P-H Patterns and Predictions on multiple input files a *response file* option has been added to the following commands *csv2dat*, *ustruct2dat*, *ustruct2hcs*, and *weather2dat*.
- A new command, *count*, for creating HUGIN case and data files with word frequencies from word files has been added to the suite of tools.
- The implementation of predictive ranges has been revised to extend existing intervals to include infinity as opposed to adding new intervals.
- A cross platform Graphical User Interface (GUI) has been added to simplify the initial user experience. Basic features are included in the GUI, with the addition of advanced features planned. See separate documentation for instructions.

## Overview of new features in P-H Patterns and Predictions version 1.5

- Added non-optional argument to the *ph* command specifying the name of the file in which a model is stored to model construction and update commands.
- Added an option to the *ph* command to enforce no discretization of numerical variables.

## Overview of new features in P-H Patterns and Predictions version 1.6

- The *csv2dat* command now creates a file specifying all the variables included in the data file. This file can be used by the *feature* command.
- Added non-optional arguments to the *feature* command specifying the name of the file in which data on the selected variables is stored and an upper limit on the number of variables.
- A new option, *-net*, has been added to the *feature* command. This option is added to increase the efficiency of feature selection on large data sets.
- To simplify the use of P-H Patterns and Predictions on multiple input files a *response file* option has been added to the *ph -inference* command. In addition a new option, *-dat*, for formatting the output of the *ph -inference* command with the response file option as a datafile has been included.
- A new option, *-net*, has been added to the *csv2dat* command. This option is added to simplify model accuracy evaluation using *accuracy*.
- Names of discretization options changed to equi-distant (formerly uniform), supervised, and entropy (formerly unsupervised).
- The P-H Patterns and Predictions software package is now available as a 64-bit version for all platforms except Mac OS X.

## Acknowledgments

HUGIN Expert A/S

Poulin Holdings LLC

September, 2007

# Contents

<b>Preface</b>	<b>iii</b>
<b>1 Introduction</b>	<b>1</b>
<b>2 Data Preparation and Manipulation</b>	<b>3</b>
2.1 Parsing, Merging, and Reducing Data . . . . .	3
2.2 Feature Extraction . . . . .	6
2.3 Case Creation and Extraction . . . . .	7
2.4 Counting Words . . . . .	8
2.5 Pulling a Web Page . . . . .	10
<b>3 Model Construction</b>	<b>11</b>
3.1 Naive Bayes Models . . . . .	11
3.2 Tree-Augmented Naive Bayes Models . . . . .	12
3.3 Hierarchical Naive Bayes Model . . . . .	13
3.4 Model Update . . . . .	14
<b>4 Inference and Analysis</b>	<b>15</b>
4.1 Inference . . . . .	15
4.2 Value of Information Analysis . . . . .	16
4.3 Scenario-Based Sensitivity Analysis . . . . .	17
4.4 Classification Accuracy . . . . .	17

<b>A Data Samples</b>	<b>19</b>
A.1 Temperature Prediction . . . . .	19
A.2 Frozen Concentrated Orange Juice . . . . .	20
A.3 Mushroom Classification . . . . .	22
A.4 Electronic Mail Filtering . . . . .	23
A.5 Classification of Marketing Advertisements . . . . .	25
<b>B Data Normalization Samples</b>	<b>29</b>
B.1 HUGIN Data File Format . . . . .	29
B.2 Frozen Concentrated Orange Juice . . . . .	30
<b>C Frequently Asked Questions</b>	<b>33</b>
<b>Bibliography</b>	<b>37</b>
<b>Index</b>	<b>38</b>

# Chapter 1

## Introduction

P-H Patterns and Predictions is a software package for pattern discovery and prediction. The software package consists of a suite of command-line based tools that allow the user or automation environment to build and use sophisticated statistical models efficiently.

The underlying technology is based on Bayesian analysis and Bayesian networks in particular. Bayesian networks are derived from Bayes' Theorem. Bayes' Theorem allows the inferring of a future event based on the prior evidence. The theorem was discovered by Rev. Thomas Bayes (1702-1761).

The P-H Patterns and Predictions software package automates the Bayesian knowledge discovery and prediction process. The tool allows the user to quickly perform extensive analysis of both structured and unstructured data. In addition, the analysis capabilities of the tool will produce rapid conclusions.

The Bayesian knowledge discovery and prediction process can be divided into three subsequent steps:

- (1) data preparation and manipulation,
- (2) model construction,
- (3) inference and analysis.

The command-line driven tools included in the software package may be grouped according to the three steps as:

**Data preparation and manipulation** The first step in any analysis is data preparation and manipulation. The tools for constructing the statisti-

cal models assume a certain data format. The goal of the data preparation and manipulation step is to organize the data to be analyzed into a data format suitable for pattern discovery and prediction.

A number of data parsing tools for formatting both structured and unstructured data are included in the package.

**Model construction** The second step is model construction. The goal of the model construction is to construct statistical models based on the data prepared in the first step.

Tools for building Naive Bayes Models, Tree-Augmented Naive Bayes Models, and Hierarchical Naive Bayes Models are included in the package.

**Inference and Analysis** The third and last step is inference and analysis. The goal of inference and analysis is to perform inference in and analysis on the statistical models constructed in the second step. The inference and analysis is performed using data prepared in the first step.

Tools for computing the probability of events given observations, performing value of information analysis on both scenarios and variables, performing scenario-based sensitivity analysis, and for determining classification accuracy are included in the package.

This reference manual contains a number of examples that illustrates the use of the P-H Patterns and Predictions software on three different data sets. The examples may be found in [Appendix A](#). The first example illustrates the use of the tools on the problem of temperature prediction in Washington, DC based on average daily temperature measures in other US cities. The second example illustrates the use of the tools on the problem of classifying mushrooms as edible or not. The third and last example illustrates the use of the tools on the problem of electronic mail classification.

The P-H Patterns and Predictions software package contains a few special-purpose tools related to the aforementioned examples.

## Chapter 2

# Data Preparation and Manipulation

P-H Patterns and Predictions is a data mining tool. It allows the user to rapidly build various types of statistical models from data. This chapter describes the tools for data preparation and manipulation included in the software package.

The tools for data preparation and manipulation can be organized into four groups: parsing data, feature extraction, case creation and generation, and pulling web pages.

The model construction tools assume data to adhere to the format of HUGIN data files. If the data does not adhere to the required format, then data may be prepared and manipulated to adhere to the required format using the tools described in the following sections.

### 2.1 Parsing, Merging, and Reducing Data

The following tools may be used to format both structured and unstructured data in the format required by the construction tools. Data is structured

when it is specified as the content of table objects in an HTML file. Data is unstructured when it is the content of an HTML file.

- ▶ `struct2dat <html> [<output>]`

This command will parse the HTML file specified in `<html>` and extracts all embedded tables that do not contain nested tables. The tables are output in a suitable format (HUGIN data file format).

The content is either displayed on standard output or stored in a file named `<output>`.

- ▶ `ustruct2dat [<-wpunct>] <html> [<output>]`

This command will parse the HTML file specified in `<html>` removing all HTML tags. The optional argument `<-wpunct>` specifies whether or not punctuation should be extracted from the HTML file. The default behavior is to remove punctuation.

The content (typically a list of words) is either displayed on standard output or stored in a file named `<output>`.

When parsing multiple HTML files it is often simpler to apply a response file instead of parsing one file at a time.

- ▶ `ustruct2dat [<-wpunct>] @<response file>`

This command will parse the list of HTML files specified in `<response file>` removing all HTML tags from each file.

The response file specifies the list of files to be parsed. Each line in the file specifies input files and an output file (in that order). Multiple input files on the same line are parsed separately, but the contents of the files (typically lists of words) are merged and saved in the output file.

The content for each file is stored in the output file specified in the response file. If a line specifies only a single file name, then this file will be input and the output will be stored in a file with name equal to the name of the input file followed by a `txt` suffix.

- ▶ `class2dat <model> <target> <classification> <output>`

This command will create a HUGIN data set based on the variables stored in the HUGIN network specification file `<model>` and the data stored in the `<classification>` file. The classification file consists of a number of lines

where each line specifies a file name and the classification of the content of the file.

The resulting data set is stored in a file named `<output>`.

- ▶ `csv2dat <output> <missing> <input> <...>`

This command will merge a set of CSV files into a single HUGIN data file. The resulting data set is stored in a file named `<output>`. The command also creates a file named `<output.net>` containing a specification of all the nodes included in `<output>`. This is useful for subsequent feature selection.

Missing values are represented with the `<missing>` symbol (e.g. `<N/A>`, `<>false>`, `<0>`, etc.).

Instead of specifying the input files as arguments to the command, it may be simpler to apply a response file.

- ▶ `csv2dat <output> <missing> @<response file>`

The response file specifies the list of input files to be merged. Each line in the file specifies the name of an input file.

- ▶ `csv2dat -net <model> <target> <output> <missing> <input>  
<...>`

This command includes in `<output>` only the variables specified in `<model>`. This command is useful in combination with *accuracy* when test data distributed over multiple input files should be restricted to a specified `<model>`.

- ▶ `csv2dat -net <model> <target> <output> <missing> @<response  
file>`

This command includes in `<output>` only the variables specified in `<model>`.

- ▶ `fit2dat <model> <data> <missing> <output>`

This command creates a HUGIN data file fitting `<model>` from the HUGIN data file `<data>`. The resulting data set is stored in a file named `<output>`. Any nodes in `<data>` not in `<model>` are removed while any nodes in `<model>` not in `<data>` are inserted with missing values in all cases.

Missing values are represented by the <missing> symbol (e.g. <N/A>, <false>, <0>, etc.).

- ▶ weather2dat <output> <input> <...>

This command will create a HUGIN data file from the input files specified as arguments. The input files are assumed to be weather data files (see [Appendix A.1](#)). The data file is stored in a file named <output>.

When parsing multiple input files it is often simpler to apply a response file instead of specifying all files as input.

- ▶ weather2dat <output> @<response file>

This command will create a HUGIN data file from the files specified in <response file>. Each line in the response file specifies the name of a single input file.

The weather data is one of the data samples considered in [Appendix A.1](#).

## 2.2 Feature Extraction

The commands described in this section may be used to identify predictor variables (feature extraction) of classification models.

The following command may be used for feature extraction in a classification model.

- ▶ feature <name> <data> <target> <states> <significance> <max number of nodes>

This method will perform a feature selection operation based on the data stored in the <data> file.

The resulting set of predictor variables and the target variable are saved to a HUGIN network specification file named <features.net> while the corresponding data is saved to a HUGIN data file with the specified <name>.

The predictor variables are identified based on statistical tests for pair-wise independence between the <target> variable and each potential predictor variable using a significance level of <significance>. Each continuous variable in the data is discretized into at most <states> number of states prior to testing.

If a non-zero value is specified for <max number of nodes> the resulting set of predictor variables will consists of (at most) the specified number of variables. If zero is specified, all variables are included in the resulting set of predictor variables.

The following command may be used for feature extraction in a classification model when the nodes of the data set is known in advance.

- ▶ `feature -net <name> <nodes> <data> <target> <states>  
<significance>`

This method will perform a feature selection operation based on the data stored in the <data> file. Only the variables of <nodes> are included in the resulting set of predictor variables. The file <nodes> could, for instance, be the nodes file constructed by *csv2dat*.

The following command may be used for feature extraction in a Boolean classifier model.

- ▶ `class2net <target> <classification> <output>`

This command will identify a set of predictor variables based on the data stored in the <classification> file.

The resulting set of predictor variables is saved to a HUGIN network specification file named <output>.

Each variable is Boolean indicating whether or not the word represented by the variable is present.

The predictor variables are identified based on statistical tests for pair-wise independence between the <target> variable and each potential predictor variable (currently a significance level of 5% is used).

## 2.3 Case Creation and Extraction

The commands described in this section may be used to create HUGIN case files from unstructured data and a HUGIN data file.

- ▶ `ustruct2hcs <model> <target> <unstruct> <...>`

This command will parse a sequence of unstructured HTML files creating one HUGIN case file for each HTML file.

The command identifies whether or not each of the variables (except the `<target>` variable) in the model stored in `<model>` is present in the HTML file.

The case file is stored in a file with the same name as the input file, but with the suffix `.hcs` added.

When parsing multiple unstructured HTML files creating multiple HUGIN case files, it is often simpler to apply a response file instead of parsing one file at a time.

- ▶ `ustruct2hcs <model> <target> @<response file>`

This command will parse the list of HTML files specified in `<response file>` removing all HTML tags from each file creating one case file for each HTML file.

The response file specifies the list of files to be parsed. Each line in the file specifies one input file and one output file (in that order).

If a line specifies only a single file name, then this file will be input and the output will be stored in a file with name equal to the name of the input file followed by a `hcs` suffix.

- ▶ `dat2hcs <model> <data> <index>`

This command will save the case with index `<index>` in the `<data>` file to a file name `case.<index>.hcs`. Cases have indexes from zero to  $N - 1$  where  $N$  is the number of cases.

## 2.4 Counting Words

The command, `count`, determines the frequency (count) of each word in an ASCII file and associates the word frequencies with a value of a prespecified target variable. The ASCII file may, for instance, contain the (multiset of) words of a HTML page after tags have been removed.

The following command creates a HUGIN data file with one case specifying word frequencies and a value of the target variable.

- ▶ `count -dat <words> <word freqs> <target> <classification> [  
 <verbose>]`

This command determines the frequency of each word in the file `<words>` and associates the frequencies with the value of `<target>` as specified in the file `<classification>`. The file `<classification>` associates each word file with a value of `<target>`. The result is stored as a HUGIN data file in `<word freqs>`.

When determining the frequency of words in multiple files it is often simpler to apply a response file instead of processing one word file at a time.

- ▶ `count -dat @<response file> <target> <classification> [  
 <verbose>]`

This command will parse the list of files specified in `<response file>` determining the frequency of words in each file separately.

The frequency of each word in each file is stored in the associated output file as specified in the response file. If a line specifies only a single file name, then this file will be input and the output will be stored in a file with name equal to the name of the input file followed by a *dat* suffix.

Given a model and associated target variable, the following command creates a HUGIN case file specifying word frequencies and a value of the target variable.

- ▶ `count -hcs <model> <words> <target> <target value> [  
 <verbose>] [  
 <notarget>] [  
 <complete>]`

The optional argument [`<notarget>`] specifies whether or not the value of `<target>` should be included in the case file, while the optional argument [`<complete>`] specifies whether or not the case file should contain a value for all variables in the model (the default value of a variable in `<model>` representing a word not included in `<words>` is zero).

When determining the frequency of words in multiple files it is often simpler to apply a response file instead of processing one file at a time.

- ▶ `count -hcs <model> <target> @<response file> [  
 <verbose>] [  
 <notarget>] [  
 <complete>]`

This command will parse the list of files specified in `<response file>` determining the frequency of words in each file separately.

The frequency of each word in each file is stored in the associated output file as specified in the response file. If a line specifies only a single file name, then this file will be input and the output will be stored in a file with name equal to the name of the input file followed by a *hcs* suffix.

## 2.5 Pulling a Web Page

The command described in this section may be used to pull a web page.

- ▶ `pull <url> <output>`

This command will save the web page specified in the `<url>` to a file named `<output>`.

## Chapter 3

# Model Construction

P-H Patterns and Predictions allows the user to rapidly build various types of statistical models from data. This chapter describes the tools for building complex statistical models included in the software package.

The tools for model construction can be organized into four groups: construction of Naive Bayes Models, construction of Tree-Augmented Naive Bayes Models, construction of Hierarchical Naive Bayes Models, and model update.

In this chapter we assume data adheres to the HUGIN data file format. If this is not the case, then data may be prepared and manipulated using the tools described in the previous chapter. The following sections describe how to rapidly build complex statistical models from data.

### 3.1 Naive Bayes Models

The commands described in this section may be used to automatically construct Naive Bayes Model classifiers.

- ▶ `ph -nbm <name> <data> <target> <states> <iterations>`  
    [<verbose>]

This command builds a Naive Bayes Model from the data contained in the <data> file with <target> as the hypothesis variable.

All continuous variables are discretized into at most <states + 2> states. Discretized continuous variables will cover the full range from minus infinity to infinity. Three different discretization methods are supported: [<

entropy>], [`<-supervised>`], and [`<-equidistant>`]. The default method is [`<-entropy>`]. If the value of `<states>` is set to zero, then no discretization is performed.

The parameters of the model are estimated using the Expectation Maximization (EM) algorithm. The EM algorithm runs with random initial parameter assignments. The argument `<iterations>` specify the number of random restarts of the EM algorithm.

The Naive Bayes Model constructed is stored in a file name `<name>`. The file is formatted according to the HUGIN network specification language.

- ▶ `ph -bnbm <name> <net> <data> <target> <iterations>`  
[`<verbose>`]

This command builds a Boolean Naive Bayes Model from the HUGIN network file `<net>` containing the variables of the model and the data contained in the `<data>` file with `<target>` as the hypothesis variable.

Each Boolean variable indicates the presence or absence of a word. The word represented by a variable is equal to the label of the variable.

The parameters of the model are estimated using the Expectation Maximization (EM) algorithm. The EM algorithm runs with random initial parameter assignments. The argument `<iterations>` specify the number of random restarts of the EM algorithm.

The Naive Bayes Model constructed is stored in a file name `<name>`. The file is formatted according to the HUGIN network specification language.

## 3.2 Tree-Augmented Naive Bayes Models

The commands described in this section may be used to automatically construct Tree-Augmented Naive Bayes Model classifiers.

- ▶ `ph -tan <name> <data> <target> <states> <iterations>`  
[`<verbose>`]

This command builds a Tree-Augmented Naive Bayes Model from the data contained in the `<data>` file with `<target>` as the hypothesis variable.

All continuous variables are discretized into at most `<states + 2>` states. Discretized continuous variables will cover the full range from minus infinity to infinity. Three different discretization methods are supported: [`<-`

equidistant>], [`<-supervised>`], and [`<-entropy>`]. The default method is [`<-unsupervised>`]. If the value of `<states>` is set to zero, then no discretization is performed.

The parameters of the model are estimated using the Expectation Maximization (EM) algorithm. The EM algorithm runs with random initial parameter assignments. The argument `<iterations>` specify the number of random restarts of the EM algorithm.

The Tree-Augmented Naive Bayes Model constructed is stored in a file name `<name>`. The file is formatted according to the HUGIN network specification language.

- ▶ `ph -btan <name> <net> <data> <target> <iterations>`  
    [`<verbose>`]

This command builds a Boolean Tree-Augmented Naive Bayes Model from the HUGIN network file `<net>` containing the variables of the model and the data contained in the `<data>` file with `<target>` as the hypothesis variable.

Each Boolean variable indicates the presence or absence of a word. The word represented by a variable is equal to the label of the variable.

The parameters of the model are estimated using the Expectation Maximization (EM) algorithm. The EM algorithm runs with random initial parameter assignments. The argument `<iterations>` specify the number of random restarts of the EM algorithm.

The Tree-Augmented Naive Bayes Model constructed is stored in a file name `<name>`. The file is formatted according to the HUGIN network specification language.

### 3.3 Hierarchical Naive Bayes Model

The command described in this section may be used to automatically construct Hierarchical Naive Bayes Model classifiers.

- ▶ `ph -hnbm <name> <data> <target> <states> <iterations>`  
    [`<verbose>`]

This command builds a Hierarchical Naive Bayes Model from the data contained in the `<data>` file with `<target>` as the hypothesis variable<sup>1</sup>.

---

<sup>1</sup>The `<-hnbm>` option is not available in the free version of the software

All continuous variables are discretized into at most  $\langle \text{states} + 2 \rangle$  states. Discretized continuous variables will cover the full range from minus infinity to infinity. Three different discretization methods are supported: [ $\langle \text{-entropy} \rangle$ ], [ $\langle \text{-supervised} \rangle$ ], and [ $\langle \text{-equidistant} \rangle$ ]. The default method is [ $\langle \text{-entropy} \rangle$ ]. If the value of  $\langle \text{states} \rangle$  is set to zero, then no discretization is performed.

The parameters of the model are estimated using the Expectation Maximization (EM) algorithm. The EM algorithm runs with random initial parameter assignments. The argument  $\langle \text{iterations} \rangle$  specifies the number of random restarts of the EM algorithm.

The Hierarchical Naive Bayes Model constructed is stored in a file name  $\langle \text{name} \rangle$ . The file is formatted according to the HUGIN network specification language.

### 3.4 Model Update

The command described in this section may be used to automatically update a classifier model.

- ▶ `ph -update  $\langle \text{name} \rangle$   $\langle \text{data} \rangle$   $\langle \text{model} \rangle$   $\langle \text{target} \rangle$   $\langle \text{states} \rangle$   $\langle \text{iterations} \rangle$  [ $\langle \text{verbose} \rangle$ ]`

This command updates a model to include additional variables based on the data contained in  $\langle \text{data} \rangle$ . The hypothesis variable of the model stored in  $\langle \text{model} \rangle$  is specified by the  $\langle \text{target} \rangle$  argument.

Variables in the data not represented in the original model are added to the model as children of the hypothesis variable (no structure between information variables is added). The data file should contain measures on all variables (both original and new variables).

All new continuous variables are discretized into at most  $\langle \text{states} + 2 \rangle$  states. Discretized continuous variables will cover the full range from minus infinity to infinity. Three different discretization methods are supported: [ $\langle \text{-entropy} \rangle$ ], [ $\langle \text{-supervised} \rangle$ ], and [ $\langle \text{-equidistant} \rangle$ ]. The default method is [ $\langle \text{-entropy} \rangle$ ]. If the value of  $\langle \text{states} \rangle$  is set to zero, then no discretization is performed.

The Naive Bayes Model constructed is stored in a file name  $\langle \text{name} \rangle$ . The file is formatted according to the HUGIN network specification language.

## Chapter 4

# Inference and Analysis

P-H Patterns and Predictions allows the user to rapidly perform extensive analysis of complex statistical models. This chapter describes the tools for inference and analysis included in the software package.

The tools for inference and analysis may be organized into three groups: inference in models, value of information analysis, and scenario-based sensitivity analysis.

In this chapter we assume data cases to adhere to the HUGIN case file format and models to the HUGIN network file format. The methods described in this section can be applied to any type of model specified as a HUGIN network file. The functionality is not limited to the models described in the previous chapter.

### 4.1 Inference

The command described in this section may be used to perform inference in a model.

- ▶ `ph -inference <model> <target> <case> [<output>] [-verbose]`

This command performs inference in the model specified in the `<model>` argument. Inference is performed with respect to the target specified in the `<target>` argument. The `<target>` will typically be the hypothesis variable of `<model>`. The posterior distribution in `<target>` is displayed for the case stored in the file `<case>`, which is a HUGIN case file.

If the `<output>` argument is specified the output is directed to a file named `<output>`.

If the `<-verbose>` argument is specified only the label and the probability of the state with maximum probability is displayed.

- ▶ `ph -inference <model> <target> @<response file> [<output>]`  
`[-verbose] [-dat]`

This command will parse the list of HUGIN case files specified in `<response file>` performing inference on each case file separately. If the optional argument, `[-dat]`, is specified the output will be formatted as a HUGIN data file.

## 4.2 Value of Information Analysis

The commands described in this section may be used to perform value of information analysis on a model with respect to evidence scenarios and (unobserved) variables.

- ▶ `ph -voivariabes <model> <target> <case> [<output>] [-verbose]`

This command performs a value-of-information analysis on each non-observed variable in the case stored in `<case>`. The value of information analysis is performed with respect to `<target>` variable. The `<target>` variable will typically be the hypothesis variable of `<model>`.

For each unobserved variable in `<case>`, a measure of how well it predicts `<target>` is displayed. The measure is the mutual information score between the predictor variable and `<target>`.

If the `<output>` argument is specified the output is directed to a file named `<output>`.

- ▶ `ph -voicase <model> <target> <case> [<output>] [-verbose]`

This command performs a value-of-information analysis on the case stored in `<case>`. The value of information analysis is performed with respect to `<target>` variable. The `<target>` variable will typically be the hypothesis variable of `<model>`.

A measure of how well `<case>` predicts `<target>` is displayed. The measure is the entropy score of the posterior distribution over `<target>` given the evidence in `<case>`.

If the `<output>` argument is specified the output is directed to a file named `<output>`.

### 4.3 Scenario-Based Sensitivity Analysis

The commands described in this section may be used to perform scenario-based sensitivity analysis on a model.

- ▶ `ph -whatifvariables <model> <target> <case> [<output>]`

This command performs a what-if analysis on each instantiated variable in the case file `<case>` relative to the `<target>` variable. The `<target>` variable will typically be the hypothesis variable of `<model>`.

The posterior probability of each hypothesis (each state of the target variable) is displayed for each possible value of each observed variable.

If the `<output>` argument is specified the output is directed to a file named `<output>`.

- ▶ `ph -whatifvariable <model> <target> <case> <variable> [<output>]`

This command performs a what-if analysis on `<variable>` relative to the `<target>`. The `<target>` variable will typically be the hypothesis variable of `<model>`.

The posterior probability of each hypothesis (each state of the target variable) is displayed for each possible value of the indicated variable.

If the `<output>` argument is specified the output is directed to a file named `<output>`.

### 4.4 Classification Accuracy

The command described in this section may be used to analyze the performance of a classifier model.

- ▶ `accuracy <model> <target> <data> <folds> [<-labels>]`

This command analyzes the performance of the classifier model `<model>` with `<target>` as the classification variable on the data stored in `<data>`.

The performance analysis is performed by n-fold cross validation where <folds> specifies the number of folds.

The command outputs the confusion matrix and classification accuracy and error using a zero-one loss function.

If the optional argument [<-labels>] is specified the confusion matrix is output using state labels instead of state indexes.

Any data values not represented in the model are ignored and data cases are assumed to be independent and identically distributed (this implies that each fold consists of a set of sequential data cases).

# Appendix A

## Data Samples

This appendix illustrates the use of P-H Patterns and Predictions on five different sets of data.

- The first example illustrates the use of P-H Patterns and Predictions for temperature prediction in Washington, DC based on measures of the temperature in other US cities.
- The second example illustrates the use of P-H Patterns and Predictions for predicting the closing level of frozen concentrated orange juice based on temperatures in US cities.
- The third example illustrates the use of P-H Patterns and Predictions for classifying mushrooms as edible or not.
- The fourth example illustrates the use of P-H Patterns and Predictions for electronic mail classification.
- The fifth and last example illustrates the use of P-H Patterns and Predictions for classification of marketing advertisements.

The following sections describe each example in turn.

### A.1 Temperature Prediction

The challenge is to build a (simple) model for predicting the temperature in Washington, DC based on measures of the temperatures in other US cities.

The source data is from the Global Summary of the Day database archived by the National Climatic Data Center<sup>1</sup>. Temperatures are measured in Fahrenheit.

The following walk-through illustrates the use of the P-H Patterns and Predictions on the weather data sample:

(1) Create a HUGIN data file from the structured data files containing average daily temperatures in a number of cities:

- ▶ `weather2dat weather.dat MDWASHDC.txt AKFAIRBA.txt NYNEWYOR.txt TXDALLAS.txt`

Usually, you would want to include in the model more than four cities.

(2) Build the Naive Bayes Classifier Model:

- ▶ `ph -nbm weather.net weather.dat MDWASHDC 10 1`

where *MDWASHDC* is the hypothesis and each continuous variable is discretized into at most ten intervals.

Next, you may compute the probability distribution over the possible temperatures in Washington, DC as follows:

- ▶ `ph -inference weather.net MDWASHDCD temperatures.hcs`

where *temperatures.hcs* is the HUGIN case with temperatures on a subset of the cities in the model and *weather.net* is the name of the HUGIN specification file containing the model.

This example illustrates how to build a Naive Bayes Classifier from a sample of structured data.

## A.2 Frozen Concentrated Orange Juice

The challenge is to build a model for predicting the daily frozen concentrated orange juice (FCOJ) contract close level based on measures of the temperatures in a number of US cities.

The source data is from the Global Summary of the Day database archived by the National Climatic Data Center<sup>2</sup> and the New York Board of Trade.

<sup>1</sup>The data is available from the Average Daily Temperature Archive at The University of Dayton <http://www.engr.udayton.edu/weather/>.

<sup>2</sup>The data is available from the Average Daily Temperature Archive at The University of Dayton <http://www.engr.udayton.edu/weather/>.

The FCOJ indicator used in this example is the price quotation in cents and hundredths of a cent per pound solid while temperatures are measured in Fahrenheit.

The following walk-through illustrates the use of the POULIN HUGIN Automation Tool on the weather data and FCOJ samples:

(1) Create a HUGIN data file from the structured data files containing average daily temperatures in a number of cities and a structure data file containing the closing level of FCOJ:

- ▶ `weather2dat weather_fcoj.dat MDWASHDC.txt AKFAIRBA.txt NYNEWYOR.txt TXDALLAS.txt fcoj.txt`

Usually, you would want to include in the model more than four cities.

(2) Build the Naive Bayes Classifier Model:

- ▶ `ph -nbm fcoj.net weather_fcoj.dat FCOJ 10 1`

where *FCOJ* is the hypothesis and each continuous variable will be discretized into at most ten intervals.

Next, you may compute the probability distribution over the possible FCOJ closing levels as follows:

- ▶ `ph -inference fcoj.net FCOJ temperatures.hcs`

where *temperatures.hcs* is the HUGIN case with temperatures on a subset of the cities in the model and *fcoj.net* is the name of the HUGIN specification file containing the model.

The most informative city with respect to predicting FCOJ closing level based on average daily temperatures can be identified using the value of information analysis functionality of P-H Patterns and Predictions as follows:

- ▶ `ph -voivariabes fcoj.net FCOJ temperatures.hcs`

where *temperatures.hcs* is the HUGIN case with temperatures on a subset of the cities in the model and *fcoj.net* is the name of the HUGIN specification file containing the model. The command will output a list of (city name, value) pairs sorted according to the value. The city with the highest value is the most informative city.

The impact of changes in the average daily temperature in a single city can be identified using the sensitivity analysis functionality of P-H Patterns and Predictions as follows:

- ▶ `ph -whatifvariable fcoj.net FCOJ temperatures.hcs MDWASHDC`

where *temperatures.hcs* is the HUGIN case with temperatures on a subset of the cities in the model and *fcoj.net* is the name of the HUGIN specification file containing the model. The command will output the content of three dimensional matrix of values<sup>3</sup>. The first dimension specifies the values of FCOJ, the second dimension specifies the values of MDWASHDC, and the third dimension is the probability of FCOJ given MDWASHDC.

Completed model included in Professional/Enterprise Editions.

### A.3 Mushroom Classification

The challenge is to build a model for classifying a mushroom as edible or poisonous based on various observations on the mushroom such as size and odor.

The source data is the Mushrooms Database [1] from the UCI Machine Learning Repository<sup>4</sup>.

The following walk-through illustrates the use of the P-H Patterns and Predictions on the Mushrooms Database:

- (1) Build the Tree-Augmented Naive Bayes Classifier Model:

- ▶ `ph -tan mushroom.net mushroom.dat class 2 1`

where *mushroom.dat* is the HUGIN data file containing the training set and *class* is the name of the hypothesis variable. As the data does not contain any continuous variables, the 2 argument is ignored.

---

<sup>3</sup>The best way to interpret the numbers is using a graphical visualization program that can make 3D plots

<sup>4</sup>The Mushrooms Database may be found at <ftp://ftp.ics.uci.edu/pub/machine-learning-databases/mushroom>

Next, you may compute the probability of a mushroom being edible or poisonous as follows:

- ▶ `ph -inference mushroom.net class amushroom.hcs`

where *amushroom.hcs* is the HUGIN case file containing the case to be classified and *mushroom.net* is the name of the HUGIN specification file containing the model.

The performance of the model as tool for classifying mushrooms can be assessed as follows:

- ▶ `accuracy mushroom.net mushroom.csv class 10`

where *mushroom.net* is the model structure, *mushroom.csv* is the data file, *class* the hypothesis variable, and 10 specifies the number of folds.

This example illustrates how to build a Tree-Augmented Naive Bayes Classifier from a sample of structured data.

## A.4 Electronic Mail Filtering

The challenge is to build a model for assessing whether or not a mail is spam or ham based on the words used in the mail.

The source data is constructed from a set of spam mails and mails sent to the UAI Mailing List<sup>5</sup>.

The following walk-through illustrates the use of the P-H Patterns and Predictions on the mail data sample:

- (1) Extract information from the unstructured data using data preparation functionality:

- ▶ `ustruct2dat mail1.html mail1.txt`

This needs to be done for each file of unstructured data. Each file will correspond to one mail.

- (2) Create a classification file for the extracted data specifying each data case as either *ham* or *spam*. For instance *mail1.txt spam*.

---

<sup>5</sup>See Association for Uncertainty in Artificial Intelligence: <http://www.auai.org>

(3) Identify the set of predictor variables (feature extraction):

- ▶ `class2net class classification.txt variables.net`

where *classification.txt* is the classification file created in the previous step.

(4) Prepare a HUGIN data file for building the Boolean Naive Bayes Classifier model:

- ▶ `class2dat variables.net class classification.txt classification.dat`

(5) Build the Boolean Naive Bayes Classifier Model:

- ▶ `ph -bnbm mail.net variables.net classification.dat class 1`

where *variables.net* is the set of predictor variables and the target, *classification.dat* is the data file created in the previous step, and 1 specifies the number of parameter learning restarts.

The above steps will build a Boolean Naive Bayes Classifier Model which may be used to classify new mails as either ham or spam. In order to apply the classifier on a mail, you need to construct a HUGIN case file representation of the mail (assumed to be save in a file named *mail1.txt*):

- ▶ `ustruct2hcs mail.net class mail1.txt`

where *mail.net* is the file containing the model specification.

The performance of the model as tool for classifying mails can be assessed as follows:

- ▶ `accuracy mail.net classification.dat class 10`

where *mail.net* is the model structure, *classification.dat* is the data file, *class* the hypothesis variable, and 10 specifies the number of folds.

Next, you may compute the probability of the mail being *ham* or *spam* as follows:

- ▶ `ph -inference mail.net class anewmail.hcs`

where *anewmail.hcs* is the HUGIN case file containing the case to be classified.

This example illustrates how to build a Boolean Naive Bayes Classifier from a sample of unstructured data.

## A.5 Classification of Marketing Advertisements

The challenge is to construct a model for classifying marketing advertisements into a set of predefined groups depending on the wording of the advertisements.

The data source for constructing the model consists of a set of files each file being a marketing advertisement. The set of advertisements are grouped into six different groups.

Based on the files in the data source and classification of each advertisement, we construct a HUGIN data file over word occurrences and the predefined classification. In this process the following steps are performed:

- (1) Extract the list of words from each advertisement file:

- ▶ `ustruct2dat @words_response`

where *words\_response* is a response file specifying the list of files containing the marketing advertisements. Each line in the file should specify the name of the file containing an advertisement. This file may, for instance, be created using commands like `ls offer_* >words_response` or `dir /B offer_* >words_response`.

The above command creates one word file for each advertisement file specified in the response file. Each word file will have a name equal to the name of the advertisement file followed by a *txt* suffix.

- (2) Count the number of occurrences of each word in each advertisement file and associate a classification with each advertisement file. This produces a HUGIN data file for each advertisement file containing word occurrences and the classification of the advertisement:

- ▶ `count -dat @wordFreqs_response class_variables classification`

where *wordFreqs\_response* is a response file specifying the list of word files created in the previous step, *class\_variable* specifies the name of the class variable, and *classification* is a file specifying the classification corresponding to each word file. The response file may, for instance, be created using commands like `ls offer_*.txt >wordFreqs_response` or `dir /B offer_*.txt >wordFreqs_response`. The file *classification* specifies the name of the HUGIN data file to be produced and the classification corresponding to this file. The classification file must be produced manually.

The above command creates one HUGIN data file for each word file specified in the response file. Each data file will have a name equal to the name of the word file followed by an *dat* suffix.

(3) Merge the set of HUGIN data files into a single HUGIN data file:

- ▶ `csv2dat wordFreqs.dat 0 @dat_response`

where *wordFreqs.dat* specifies the name of the HUGIN data file produced, *0* specifies a missing value indicator, and *dat\_response* is a response file specifying the list of HUGIN data files created in the previous step. The response file may, for instance, be created using commands like `ls offer_*.txt.dat >dat_response` or `dir /B offer_*.txt.dat >dat_response`.

The above command creates a single HUGIN data file.

(4) Perform feature selection (optional):

- ▶ `feature features.dat wordFreqs.dat class_variable 2 0.05 0`

where *wordFreqs.dat* is the HUGIN data file produced in the previous step, *class\_variable* is the name of the class variable, *2* specifies the number of states of continuous variables (plus predictive ranges) when performing feature selection, *0.05* specifies the significance level used in feature selection, and *0* specifies the maximum number of features (zero implies no limitation).

The above command creates a single HUGIN data file called *features.dat* containing the data over the selected features and the class variable.

(5) Construct model:

- ▶ `ph -nbm nbm.net features.dat class_variable 2 10`

where *features.dat* is the HUGIN data file produced in the previous step, *class\_variable* is the name of the class variable, *2* specifies the (maximum) number of states of continuous variables (plus predictive ranges), and *10* specifies the number of restarts of the parameter estimation.

The above command creates a single HUGIN Net file containing the classification model.

The above sequence of steps creates a Naive Bayes Model that may be used to classify unseen marketing advertisements into the set of classes specified as states of the classification variable. In order to classify a given advertisement it is necessary to produce a HUGIN Case file representing the advertisement. The model classifies marketing advertisements based on word occurrences. Thus, it is necessary to determine the frequency of each word in the model given the content of the marketing advertisement.

The following sequence of steps illustrates how to prepare a set of marketing advertisements for classification using the model constructed above.

(1) Extract the list of words from each advertisement file:

- ▶ `ustruct2dat @words_response`

where *words\_response* is a response file specifying the list of files containing the marketing advertisements to be classified. Each line in the file should specify the name of the file containing an advertisement. This file may, for instance, be created using commands like `ls offer_* >words_response` or `dir /B offer_* >words_response`.

The above command creates one word file for each advertisement file specified in the response file. Each word file will have a name equal to the name of the advertisement file followed by a *txt* suffix.

(2) Create one HUGIN Case file for each word file

- ▶ `count -hcs nbm.net class_variable @hcs_response -complete -notarget`

where *nbm.net* is the HUGIN Net file containing the classification model, *hcs\_response* is a response file specifying the list of word files created in the previous step, *class\_variable* specifies the name of the class variable, *-complete* is an option specifying that none occurring words should be assigned the value zero, and *-notarget* specifies that the target variable should not be included in the case file. Each line in the file should specify the name of the file containing the words of an advertisement. This file may, for instance, be created using commands like `ls offer_*.txt >hcs_response` or `dir /B offer_*.txt >hcs_response`.

The above command creates one case file for each advertisement file specified in the response file. Each case file will have a name equal to the name of the word file followed by a *hcs* suffix.

The prediction of the model on a particular marketing advertisement is determined as follows:

- ▶ `ph -inference nbm.net class_variable offer_11.hcs`

where *nbm.net* is the HUGIN Net file containing the classification model, *class\_variable* specifies the name of the class variable and *offer\_11.hcs* specifies the HUGIN Case file representing the marketing advertisement.



## Appendix B

# Data Normalization Samples

This appendix illustrates the use of P-H Patterns and Predictions for data normalization. Data normalization is the process of organizing multiple data sources into the proper format for P-H Patterns and Predictions.

With P-H Patterns and Predictions models are constructed using the *ph* command. The *ph* command supports the construction of Naive Bayes Model, Hierarchical Naive Bayes Models, and Tree-Augmented Naive Bayes Models.

The *ph* command constructs any of the aforementioned models based on a data file of cases specified as input. The data file of cases should adhere to the HUGIN data file format (a CSV-like format). Thus, it is necessary to normalize the data sources to this format.

### B.1 HUGIN Data File Format

A HUGIN data file is a text file containing the data cases to be used for model construction and model update. A HUGIN data file contains the specification of a database of cases over a set of variables where each case specifies an assignment of a value to each variable.

The first line consists of a specification of the variables contained in the database. Each variable must have a unique name specified as a C-like identifier (a letter followed by a sequence of letters, digits, or underscores). Variable names are separated using comma.

For instance, the first line of a data file may look like this:

MDWASHDC, ALBIRMIN, AZPHOENIX, FCOJ

Each of the following lines specifies a case in the database. As such each line specifies the value of each of the variables specified in the first line of the file. For instance, the second line in the file may look like this:

40.8, 35.6, 56.4, 90.70

This line specifies that the value of MDWASHDC is 40.8, the value of ALBIRMIN is 35.6, and so on.

The value of a variable may be missing. This is indicated using the *N/A* symbol.

## B.2 Frozen Concentrated Orange Juice

Below we describe the data normalization process necessary for successfully applying the PHPP tool to the task of predicting the closing level of frozen concentrated orange juice (FCOJ) based on average daily temperatures in US cities.

There are two data sources to consider in the FCOJ example. One data source specifies average daily temperatures in US cities while the second data source specifies the closing level of frozen concentrated orange juice.

The average daily temperatures for US cities are distributed as separate data files for each city. For instance, the file MDWASHDC.txt specifies the average daily temperature in Washington, DC. The format of the data file is as indicated below:

1	1	1995	40.6
1	2	1995	39.8
1	3	1995	29.3
1	4	1995	33.0
1	5	1995	20.9
1	6	1995	27.0
...			

The data file consists of a sequence of lines where each line specifies a date and the average temperature on that date. The first line, for instance, specifies that the average daily temperature on January 1st, 1995 was 40.6 degrees.

Notice that all dates are not necessarily included in the data file for a specific city.

The daily closing level of frozen concentrated orange juice is distributed as a separate CSV-file. The format of the data file is as indicated below:

```
06/01/99  90.85
06/02/99  93.20
06/03/99  90.65
06/04/99  90.60
06/05/99  91.00
06/07/99  90.70
...
```

The data file consists of a sequence of lines where each line specifies a date and the closing level of frozen concentrated orange juice on that date. The first line, for instance, specifies that the closing level of frozen concentrated orange on June 1st, 1999 was 90.85.

In fact the closing level of frozen concentrated orange juice is distributed as a Microsoft Excel file with some additional information. It is a simple task to copy the two columns corresponding to the date and closing level to a separate CSV file though.

Notice again that all dates are not necessarily included in the data file for a specific city.

The *weather2dat* command can be applied to normalize the data sources into a single HUGIN data file. The *weather2dat* takes as input a sequence of files in the format of the weather data. This implies that it is necessary to transform the frozen concentrated orange juice data file into a file adhering to the weather file format. This can be done in Microsoft Excel using functions for extracting day, month, and year of a date. This will produce a file with the format as indicated below:

```
6      1      1999  90.85
6      2      1999  93.20
6      3      1999  90.65
6      4      1999  90.80
6      5      1999  91.00
6      6      1999  90.70
...
```

Once the data normalization of input files has been completed, the files are merged into a single HUGIN data file using the *weather2dat* command as follows:

- ▶ `weather2dat weather.dat MDWASHDC.txt FCOJ.txt`

The output file *weather.dat* has the format:

```
MDWASHDC, FCOJ
40.6, N/A
39.8, N/A
29.3, N/A
33.0, N/A
20.9, N/A
27.0, N/A
...
77.5, 90.85
78.6, 93.20
78.5, 90.65
71.2, 90.60
72.5, 91.00
73.8, N/A
83.1, 90.70
...
```

where *N/A* indicates a missing data value. Notice that the starting date of the FCOJ file is June 6th, 1999 while the starting date of the weather file is January 1st, 1995. Thus, the value of FCOJ is missing for a large number of cases in the beginning of the file.

The data file constructed may be used to construct models using the *ph* command. Including the average daily temperature for additional cities is done simply by adding the corresponding data files to the input to the *weather2dat* command.

## Appendix C

# Frequently Asked Questions

This appendix gives answers to frequently asked questions on the use of the P-H Patterns and Predictions software package. In this appendix we refer to P-H Patterns and Predictions as PHPP.

(1) **Question: How do I avoid discretization of numerical data?**

**Answer:** There are two possible solutions. One solution is to relabel the data entries with numerical values into words.

For instance, the data set contains numerical values specifying months and the days of the month as an integer, i.e. the month is indicated as a number  $1, 2, \dots, 12$  and the day of the month is indicated as a number  $1, 2, \dots, 31$ . To avoid discretization of day and month the values can be relabeled into words: *First*, *Second*, and so on for days and *January*, *February*, and so on for months.

The other solution is to use the `<ph -update>` command on a HUGIN network file that contains variables representing day and month. That is, a model with two variables is created using the HUGIN GUI. One numbered variable with states  $1, \dots, 12$  representing the month and one numbered variable with states  $1, \dots, 31$  representing the day of the month are created. The model is store in a HUGIN network specification file. This file is given as an argument to `<ph -update>` command along with the initial data set.

(2) **Question: Why does PHPP create only 3 intervals when I specify that it should create 5?**

**Answer:** The PHPP assumes that the data set is a representative sample of the underlying process generating the data. Hence, the data set contains all representative sample of possible values for numerical valued entities.

The PHPP creates as many intervals as specified by the user with the number of different values in the data set being an upper limit.

To increase (or decrease) the number of intervals created by PHPP the HUGIN GUI can be used for post-processing of the constructed model. The HUGIN GUI can also be used to create the necessary intervals for selected variables prior to model construction. This can be achieved using the <ph -update> command as described under Q&A (1).

(3) **Question: What discretization processes are implemented?**

**Answer:** Uniform (equal distance), supervised (information gain), and unsupervised (equal frequency) discretization.

(4) **Question: How does PHPP support unsupervised discretization?**

**Answer:** The unsupervised (equal frequency) discretization process aims at creating intervals such that each interval contains  $N/n$  values where  $n$  is the number of intervals and  $N$  is the number of cases. The aim is to create a uniform prior distribution on the continuous variable.

First the values are sorted in increasing order. Secondly the algorithm works from low to high values splitting an interval on the average between two neighbor values belonging to two different intervals. Each interval is created by counting values from low to high and specifying the end point of the interval after  $N/n$  values. The end point of the current interval (which is the start point of the subsequent interval) is determined as the average of the last value in the current interval and the first value of the subsequent interval.

(5) **Question: How does PHPP support supervised discretization?**

**Answer:** The supervised (information gain) discretization process implements a method based on [2].

(6) **Question: How do I extract a HUGIN case file for a particular case from a HUGIN data file?**

**Answer:** If you would like to construct a HUGIN case file for a particular case in a HUGIN data file, then you can use the <dat2hcs> command. Assume you have a HUGIN data file named <datafile.csv> with 10000 cases and would like to construct a HUGIN case file named <case\_10.hcs> for the case with index 10 in <datafile.csv>.

The <dat2hcs> takes as argument a model that is constructed from <datafile.csv>. The command is

► `dat2hcs <model.net> <datafile.csv> <10>`

where <model.net> contains the model.

- (7) **Question: What is the command for evaluating the performance of a PHPP model?**

**Answer:** PHPP version 1.6 implements n-fold cross validation.

The proper algorithm to use for evaluating the performance of a model may, however, depend on the domain in which the model is to be used and what the model is to be used for.

Additional algorithms to consider for model evaluation are receiver operating characteristic curves (ROC) and area-under-curve of ROC to name a few. Later versions of PHPP will implement such algorithms.

- (8) **Question: Are characters such as ",", ".", ";", etc ignored?**

**Answer:** Characters such as ",", ".", ";", etc are ignored when creating a Boolean model from unstructured data as in the example of mail classification unless the <-wpunct> option is specified. In this case the aforementioned characters are assumed to be part of the word preceding the character.

White spaces should be used as word separators when building a Boolean model.

- (9) **Question: What kind of models does PHPP construct?**

**Answer:** PHPP supports the construction of Naive Bayes, Tree Augmented Naive Bayes models, Hierarchical Naive Bayes Models.

Using PHPP it is possible to construct models containing a mixture of Boolean, qualitative and quantitative variables.

Each model will have a unique target variable. The target variable may be considered as the hypothesis variable or classification variable when the model is to be used for classification. The target variable may have multiple states.

- (10) **Question: How do I find the most informative case?**

**Answer:** The most informative case is the case which reduces the entropy of the target variable the most. The entropy may be interpreted as a measure of the degree of chaos in a probability distribution. Since a PHPP model contains a unique target variable, the most informative case is the case which reduces the entropy of the posterior distribution of the target variable the most.

The <ph -voicase> command computes a measure of how well a single case predicts the target variable. The measure computed by this command is referred to as the value-of-information score of the case. The higher the score, the better the case predicts the target variable. Finding the most informative case is then a question of identifying the case with the highest value-of-information score.

(11) **Question: Can I apply PHPP on a partly specified model structure?**

**Answer:** A model constructed with the HUGIN GUI and saved as a HUGIN network file can be given as input to PHPP using the <ph -update> command.

(12) **Question: How do I display the model?**

**Answer:** The HUGIN GUI may be used to display a model constructed using PHPP. PHPP uses the underlying HUGIN Decision Engine as its core engine. The models built by PHPP are stored in HUGIN network files.

(13) **Question: How do I make adjustments to a PHPP model?**

**Answer:** The HUGIN GUI may be used to adjust a model constructed using PHPP. PHPP uses the underlying HUGIN Decision Engine as its core engine. The models built by PHPP are stored in HUGIN network files.

In addition the <ph -update> allows the user to update a model based on a new set of data.

(14) **Question: Can PHPP handle different types of data in the same model?**

**Answer:** Boolean, continuous and discrete valued entities may be represented in the data set from which a model is constructed.

The PHPP will represent a continuous valued entity using discretization. The discretization process is explained in Q&As (1) – (3).

(15) **Question: Does PHPP support database connectivity?**

**Answer:** Yes. The README file distributed with the installation specifies how to use the database connectivity of PHPP.

# Bibliography

- [1] C. Blake and C. Merz. UCI repository of machine learning databases, 1998.
- [2] U. M. Fayyad and K. B. Irani. Multi-Interval discretization of continuous valued attributes. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence*, pages 1022–1027, 1993.

# Index

*abc*, 32  
*accuracy*, 17  
  
*class2dat*, 4  
*class2net*, 6  
*count -dat*, 8  
*count -dat with response file*, 8  
*count -hcs*, 8  
*count -hcs with response file*, 9  
*csv2dat*, 5  
*csv2dat with response file*, 5  
  
*dat2hcs*, 7  
  
*feature*, 6  
*fit2net*, 5  
  
*ph -bnbm*, 12  
*ph -btan*, 13  
*ph -hnbm*, 13  
*ph -inference*, 15  
*ph -nbm*, 11  
*ph -tan*, 12  
*ph -update*, 14  
*ph -voicase*, 16  
*ph -voivariables*, 16  
*ph -whatifvariable*, 17  
*ph -whatifvariables*, 17  
*pull*, 9  
  
*struct2dat*, 4  
  
*ustruct2dat*, 4  
*ustruct2dat with response file*, 4  
*ustruct2hcs*, 7  
*ustruct2hcs with response file*, 7  
  
*weather2dat*, 5  
*weather2dat with response file*, 5