“Exploiting relational database technology for statistical machine learning in FactorBase”

Master Thesis Software Engineering
Faculty of Management, Science & Technology

Student: Rene Quakkelaar
Student number: 838975054
Institute: Open University, the Netherlands
Master Program: Software Engineering
Course: IM9906
Date: July 6, 2017
Chair-man: Dr. Bastiaan Heeren
Supervisor: Dr. Arjen Hommersom
Acknowledgements

After an intensive period of one year, today is the day: writing this note of thanks is the finishing touch on my thesis. It has been a period of learning for me, not only in the scientific area but also on a personal level. Writing this thesis had a big impact on the time I did spend last year. I would like to reflect on the people who have supported and helped me throughout this period.

I would first like to thank my supervisor Dr. Arjen Hommersom for his feedback, enthusiasm and his valuable guidance. He definitely provided me with the right tools that I needed to choose the right direction and successfully complete my thesis.

I would also like to thank my family and especially my wife for their wise council and sympathetic ear and for being the greatest supports one could ever wish for.
Abstract

Nowadays many relational databases store data in huge quantities in a relational format, with different tables for entities and information about the links between those entities. The field of Statistical Relational Learning has developed various statistical models for this type of relational data. Such a multi-relational statistical model provides an integrated analysis of the heterogeneous and interdependent data in the database. FactorBase is an SQL based framework that leverages a relational database management system to support multi-relational model discovery. It stores and manages the statistical models inside the database together with the data used. FactorBase supports learning a first-order Bayesian network model for the entire database using SQL.

In this thesis we want to research if we can exploit relational database technology to enhance and improve the FactorBase algorithms that are used to derive statistics for learning the structure of the network and its parameters. We propose a new algorithm that dynamically builds views that represent the statistics needed for learning the structure of the database. Leveraging various database features like nested views, prepared statements and utilizing the power of the database engine we are able to dynamically build these views that only have to be created once in a very short timeframe. Using these views, the data can be changed dynamically, reflecting these changes in the statistics needed for learning the network structure. The second enhancement proposed is an algorithm that improves the time needed to calculate column specific statistics like data value frequency and likelihood for the parameter learning phase executed by FactorBase. This new algorithm also greatly reduces the number of statements needed to generate the statistics for each column.

The new algorithms and logic are compared in time and resources needed against a baseline running FactorBase that is conducted using six publically available data sets. The results show an improvement in time and resources needed when applying the new algorithms, reducing the time needed to discover the best model for the data being learned from. Future work is needed to implement these new algorithms in a future FactorBase version allowing bigger data sets to be examined in shorter period of time and using less resources.
# Table of contents

**Illustration index** ..................................................................................................................................................... 6

1. **Introduction** .................................................................................................................................................... 8  
   1.1 Motivation ......................................................................................................................................................... 9  
   1.2 Problem ........................................................................................................................................................... 9  
   1.3 Thesis overview and organization .................................................................................................................. 10

2. **Background** ..................................................................................................................................................... 11  
   2.1 Basic concepts of probability .......................................................................................................................... 11  
   2.2 Random variables ............................................................................................................................................ 12  
   2.3 Conditional probability and random variables ............................................................................................. 12  
   2.4 Probabilistic graphical models ...................................................................................................................... 13  
      2.4.1 Bayesian Network ....................................................................................................................................... 14  
      2.4.2 Markov Network ....................................................................................................................................... 17  
      2.4.3 Factor graphs ........................................................................................................................................... 19  
      2.4.4 Markov blanket ........................................................................................................................................ 20  
      2.4.5 Bayesian Networks versus Markov Networks ......................................................................................... 21  
      2.4.6 Inference .................................................................................................................................................. 21  
      2.4.7 Learning .................................................................................................................................................. 26  
   2.5 Multi-relational learning .................................................................................................................................. 27  
      2.5.1 Background .............................................................................................................................................. 27  
      2.5.2 Example .................................................................................................................................................... 28  
      2.5.3 Multi-relational learning tools .................................................................................................................. 29  
   2.6 FactorBase, a high-level overview ................................................................................................................... 30  
   2.7 Relational databases ........................................................................................................................................ 35  
      2.7.1 Views in a relational database .................................................................................................................. 35  
      2.7.2 Prepared statements .................................................................................................................................. 36  
      2.7.3 Stored procedures .................................................................................................................................... 38

3. **Research design** .................................................................................................................................................. 39  
   3.1 Research questions ......................................................................................................................................... 40  
   3.2 Research scope .............................................................................................................................................. 40  
   3.3 Research validation ......................................................................................................................................... 41

4. **FactorBase: Learning multi-relational graphical model example** ................................................................. 42  
   4.1 Unielwin entity relationship diagram .............................................................................................................. 42  
   4.2 Parameterized factors ...................................................................................................................................... 42  
   4.3 CT generation .................................................................................................................................................. 43  
   4.4 CPT generation .............................................................................................................................................. 47

5. **Dynamic Views for learning from multi-relational data** .................................................................................. 50  
   5.1 Dynamic views ................................................................................................................................................ 50  
   5.2 What to create dynamic views from ................................................................................................................ 50  
   5.3 Advantages of views over base tables in FactorBase ..................................................................................... 51  
   5.4 Creating dynamic views using prepared statements .................................................................................... 51  
   5.5 Implemented framework .................................................................................................................................. 54  
      5.5.1 Framework design ....................................................................................................................................... 54  
      5.5.2 Dynamic view algorithm for CTs ............................................................................................................... 55  
      5.5.3 Dynamic view algorithm for CPTs ............................................................................................................. 56  
      5.5.4 Framework validation .................................................................................................................................. 57

6. **Reducing data movement and query complexity when learning from multi-relational data** .................. 60
Exploiting relational database technology for statistical machine learning in FactorBase

6.1 Building models inside the database ...............................................................60
6.2 Data movement and query reduction algorithm ........................................62
6.3 Implemented framework ............................................................................62
6.4 Algorithm example ......................................................................................62

7. Experiments ...................................................................................................66
7.1 Baseline test FactorBase ............................................................................66
7.1.1 Datasets ....................................................................................................66
7.1.2 Test environment ....................................................................................66
7.1.3 Test setup ................................................................................................67
7.1.4 Test execution ..........................................................................................67
7.1.5 Conclusions ............................................................................................68
7.2 Dynamic views for learning from relational data ......................................69
7.2.1 Experiment results ..................................................................................69
7.2.2 Conclusions .............................................................................................72
7.3 Reducing data movement and query complexity when learning from relational data ....73
7.3.1 First experiment on a small data set ..................................................74
7.3.2 Second experiment on a large data set ..............................................75
7.3.3 Conclusions ............................................................................................77

8. Conclusion ......................................................................................................79
8.1 Answers to research questions ..................................................................79
8.2 Limitations ..................................................................................................80
8.2.1 Research related .....................................................................................80
8.2.2 Software related .....................................................................................80
8.3 Contributions ...............................................................................................81
8.4 Future work ................................................................................................82

9. Bibliography ..................................................................................................83

10. SQL results and program output Index .....................................................85

11. Appendix .....................................................................................................86
11.1 Running the unielwin dataset from the terminal command line using FactorBase 86
11.2 Baseline test environment .......................................................................87
11.3 Baseline results FactorBase per dataset ..............................................88
11.3.1 Financial_std dataset ..........................................................................88
11.3.2 Hepatitis_std dataset ..........................................................................91
11.3.3 MovieLens_std dataset .......................................................................93
11.3.4 MovieLens_TQ dataset ......................................................................95
11.3.5 Mutagenesis_std dataset ...................................................................97
11.3.6 Unielwin dataset ................................................................................99
11.4 Dynamic view testing results ................................................................101
11.4.1 Financial_std dataset ..........................................................................101
11.4.2 Hepatitis_std dataset ..........................................................................102
11.4.3 MovieLens_std dataset ......................................................................103
11.4.4 MovieLens_TQ dataset ......................................................................104
11.4.5 Mutagenesis_std dataset ....................................................................105
11.4.6 Unielwin dataset ...............................................................................106
11.5 Source code samples ...............................................................................107

5
Illustration index

Figure 2.1 A Bayesian Network with five nodes ................................................................. 14
Figure 2.2 An example Bayesian Network with five nodes ............................................. 16
Figure 2.3 A Bayesian Network with five nodes and CPT nodes ................................. 16
Figure 2.4 A Markov Network with five nodes ................................................................. 17
Figure 2.5 A Markov Network with six nodes .................................................................. 18
Figure 2.6 Markov Network properties ............................................................................. 18
Figure 2.7 A factor graph with corresponding Bayesian Network .................................... 19
Figure 2.8 A factor graph with corresponding Markov Network ...................................... 20
Figure 2.9 A Markov Blanket for node N in a Bayesian Network ...................................... 20
Figure 2.10 Example conversion of a Bayesian Network to a Markov Network .............. 21
Figure 2.11 Example of directed network with five nodes ............................................... 23
Figure 2.12 An example network with three nodes .......................................................... 24
Figure 2.13 Example factor graph with features ............................................................... 24
Figure 2.14 Parameter learning example ........................................................................... 26
Figure 2.15 Structure learning example ............................................................................ 27
Figure 2.16 A database representation of three tables storing accidents ......................... 28
Figure 2.17 A generic design for a SRL solution for learning multi-relational data ......... 29
Figure 2.18 The overall system flow in FactorBase with the three components ............... 31
Figure 2.19 Representation of the Order related tables with data .................................... 32
Figure 2.20 Partial relational Bayesian Network for the OrderedBooks relation ............. 32
Figure 2.21 CPT and CT example for a column from the table OrderedBooks ............. 33
Figure 2.22 The SQL for the creation of a CT for a list of variables ................................ 34
Figure 2.23 The SQL used to determine the CPT for a variable ....................................... 34
Figure 2.24 SQL example for creation of a database view ................................................ 35
Figure 2.25 The anatomy of a database view pictured ....................................................... 36
Figure 2.26 MySQL statements needed to use prepared statements ............................... 36
Figure 2.27 SQL query example for a prepared statement ............................................... 37
Figure 2.28 Example SQL query for a prepared statement using a SQL string .............. 37
Figure 2.29 Example SQL syntax to create a stored procedure ....................................... 38
Figure 2.30 Example SQL to execute a stored procedure ............................................... 38
Figure 3.1 Research model used ....................................................................................... 39
Figure 4.1 The unielwin data model in an Entity Relation Diagram .................................. 42
Figure 4.2 The SQL for the creation of a CT for a list of variables ................................... 43
Figure 4.3 SQL query to generate the CT for the relation prof in the unielwin dataset ... 44
Figure 4.4 Generic SQL query to generate a CT for a relationship par-RV ...................... 44
Figure 4.5 SQL query to generate a CT for relation prof-student in the unielwin dataset .. 44
Figure 4.6 First step for generating the CT for the relation prof-student in the unielwin dataset ............................................................................................................ 45
Figure 4.7 Second step for generating the CT for the relation prof-student in the unielwin dataset ............................................................................................................ 46
Figure 4.8 Final step for generating the CT for the relation prof-student in the unielwin dataset ............................................................................................................ 47
Figure 4.9 Example SQL query for generating the CPT for the relation capability of prof-student in the unielwin dataset ........................................................................ 48
Figure 5.1 FactorBase and the concept of .csv files ......................................................... 50
Exploiting relational database technology for statistical machine learning in FactorBase

Figure 5.2 Example SQL that generates a SQL string for a CREATE VIEW statement .......... 52
Figure 5.3 The result SQL query that contains the CREATE VIEW statement .................. 52
Figure 5.4 Implementation overview of the dynamic view framework ......................... 54
Figure 5.5 Dynamic view algorithm for CT creation in a database ............................. 55
Figure 5.6 Dynamic view algorithm for CPT generation in a database ....................... 56
Figure 6.1 Dynamic view algorithm for reducing data movement and reducing query complexity when calculating column frequencies ............................................. 62
Figure 7.1 Baseline datasets used for testing ................................................................. 66
Figure 7.2 Baseline results for all datasets measured in milliseconds ....................... 68
Figure 7.3 Overview of percentage time used for CT and CPT generation for all datasets used .................................................................................................................. 69
Figure 7.4 Example output of a MySQL performance report ........................................ 70
Figure 7.5 CT building time comparison FactorBase versus dynamic views .................. 71
Figure 7.6 Dataset properties overview ..................................................................... 71
Figure 7.7 Parameter learning time comparison between FactorBase and dynamic views ... 72
Figure 7.8 MySQL output showing properties of the table used .................................. 74
Figure 7.9 I/O's compared between baseline query and query using new algorithm ........ 74
Figure 7.10 Execution time comparison between baseline and new algorithm ............ 75
Figure 7.11 MySQL table properties for the large data test set .................................... 75
Figure 7.12 Comparison of I/O’s needed for baseline and new algorithm for large dataset .. 76
Figure 7.13 Query execution time difference between baseline and new algorithm ......... 77
1. Introduction

Advances in the information technology allow researchers and the industry community to store huge quantities of data in digital format. In the modern age, virtually all automated systems generate some form of data either for diagnostic or analytic purposes. The data volumes are, however, such that there is often more data than what can really be processed using traditional techniques. In other words, all this data is of no value without mechanisms to efficiently and effectively extract information and knowledge from it. This fact has resulted in a new research field known as ‘Knowledge Discovery in Databases’ (Knobbe 1999), often referred to as ‘Data Mining’. At the same time there is a different research field called ‘Machine Learning’ that evolved from the study of pattern recognition and computational learning theory in artificial intelligence. Machine learning focuses on performing a specific task, such as prediction, based on a model learned from data. Data Mining on the other hand focuses on the discovery of unknown patterns in the data. The two research fields overlap in many ways. Data Mining uses a lot of the Machine Learning methods but often for a different purpose. Machine Learning will also employ Data Mining methods, for example for unsupervised learning\footnote{Unsupervised systems are not provided any training examples and conduct data clustering. This is the division of data instances into several groups. The results of the clustering algorithms are data driven, hence more ‘natural’ and better suited to the underlying structure of the data. This advantage is at the same time a major drawback. Without a possibility to tell the machine what to do (such as in classification i.e. supervised learning), it is difficult to judge the quality of clustering results in a conclusive way. The absence of training example preparation makes the unsupervised paradigm very attractive.}. In this research proposal we will focus on Machine Learning when looking at learning a model from data stored in a relational database. Where needed we will explicitly mention which research field of interest is applicable.

Until recently a lot of the research in both previously mentioned research fields used models and mechanisms based on data being stored in a single table or data-matrix. One of the exiting new research areas within knowledge discovery whereby data is stored in multiple tables is how to leverage database systems to support multi-relational model discovery. Model learning in this context is about examining a (potentially) large number of statistical associations across multiple tables in a database, and based on that, building models from it. In this thesis we would like to focus on enhancing FactorBase (Olivier Schulte 2015), an existing multi-relational framework for model learning.

---

8
1.1 Motivation
The intersection of database-management and machine learning is heavily researched at this moment. Database researchers did notice the usefulness of statistical relational models for knowledge discovery and representing uncertainty in databases (Wang 2008). However, often these models focus on inference\(^2\) given a statistical relational model, not on learning the model itself from the data stored. A different approach is FactorBase, a Structured Query Language (SQL) based framework to leverage database management systems to support multi-relational model discovery (i.e. learning a model). With FactorBase the graphical representation of the model is pushed inside the database allowing the researchers to use SQL as a high-level scripting language for Statistical Relational Learning (SRL) based on multi-relational data. Experiments show significant scalability gains with respect to the number of records researched and the ability to discover more complex cross-table relations than previous SRL based systems such as BayesStore and Tuffy (Niu 2011).

1.2 Problem
There have been tremendous improvements in techniques for collecting, storing and transferring large volumes of data. Together with these improvements there was a dramatic increase in the amount of data, typically stored in relational database systems. There is a constant demand for better and improved techniques to keep up with the growing amount of data that needs to be explored.

In this thesis we would like to explore if the SQL (and underlying algorithms) used in FactorBase can be improved and if a different database management engine (in memory, column versus row store) can be used to push the boundaries with respect to performance and scalability further. This is combined with a limited literature study to support the experiments conducted. We continue based on the conclusion and future work as described by Schulte et al. (Olivier Schulte 2015, page 10) for the FactorBase SQL framework. They describe several opportunities for optimizing the relational database operations. These include the SQL used, various ways of storing statistics, materializing table views, using main memory versus disk etc. To keep up with the increasing size of data volumes in relational databases, further research is justified to improve learning from multi-relational data. The research in this area is important for database researchers too. Most database systems do support data mining and the increasing volumes and complexity is challenging. The results of the research using FactorBase can therefore also be relevant for database research in general.

\(^2\) Inference in this context is defined as deriving rational conclusions from premises known or assumed to be true.
1.3 Thesis overview and organization

In chapter 2 we provide a theoretical background on probability, random variables, (probabilistic) graphical models, relational databases and a description of a high level overview of FactorBase. In chapter 3 we outline the research-questions and –scope together with the research approach we will follow in this thesis. To properly introduce FactorBase, chapter 4 will contain a description of a FactorBase example run for one of the data sets we will use throughout the research. In chapter 5 we introduce two new algorithms using dynamic views that improve the overall learning algorithm for learning from multi-relational data. Chapter 6 is used to outline a new algorithms used to reduce the data-movement while learning from multi-relational data and reducing query complexity at the same time. The results of the various experiments and the base line test are outlined in chapter 7, followed by the overall conclusions and future work in chapter 8.
2. Background

Many organizations today maintain their data in relational databases, which support complex structured data. Extending machine learning from traditional single-table methods to multi-relational data is an important direction for practical applications. The statistical and algorithmic challenges that arise from multi-relational data have been researched using Statistical-Relational Learning (SRL), Multi-Relational Data Mining (MRDM), and Inductive Logic Programming (ILP) (De Raedt 2008). To properly understand the proposed improvements for multi-relational learning using FactorBase, we have to define some basic concepts with respect to probability and graphical models to represent probabilistic dependencies between data attributes. In this chapter we present some theoretical background on probability, random variables, (probabilistic) graphical models, multi-relational learning, a high-level overview of FactorBase and a description of the relational database techniques we will use later on.

2.1 Basic concepts of probability

Imprecise and incomplete knowledge mitigate against predicting an event with certainty. An event is defined as an outcome or collection of outcomes, and an outcome is defined as an instance of a single experiment result. Probability theory provides a mechanism to deal with uncertainty. The probability concept of an uncertain, random event relates to measuring a chance, or likelihood, that an event will occur. This in contrast with a statistical experiment in which events are sampled from a population to make statistical inferences.

The probability $P(A)$ of an event $A$ to occur in a sample space $S$, takes a real number value from the range $[0,1]$. The probability system assigns probabilities $P(A)$ to event $A$ based on three probability axioms.

Axiom 1. The probability is greater or equal to zero and less or equal to 1:

$$0 \leq P(A) \leq 1 \text{ for each event } A \tag{1}$$

Axiom 2. This is the assumption of unit measure: the probability that at least one of the events in the sample space will occur is 1.

$$P(S) = 1 \tag{2}$$

Axiom 3. For non-overlapping events (disjoints), the probability of their union is equal to the sum of the probabilities of the individual events:

$$P(A \text{ or } B \text{ or } C \text{ or } ...) = P(A \cup B \cup C \cup ...) = P(A) + P(B) + P(C) + ... \tag{3}$$

The fact that an event already occurred may influence the occurrence of another event, and with that it influences the event of this event’s probability. A conditional probability is the probability of an event occurring given the knowledge that another event already occurred. The conditional probability that event $A$ will happen given that event $B$ has already happened is denoted by $P(A|B)$.

Conditional probability allows us to use existing knowledge that concerns events that happened prior to a given event in order to improve a probability estimation of this event.
The probability $P(A)$ of event $A$ prior to any observation is called a prior probability. A conditional probability, which is estimated after some measurements or prior events outcomes is called a posterior probability.

We define a conditional probability of event $A$ given event $B$ as:

$$P(A|B) = \frac{P(A \cap B)}{P(B)} \quad \text{with } P(B) \neq 0. \quad (4)$$

Formally we can define two events $A$ and $B$ to be independent if:

$$P(A|B) = P(A) \text{ and } P(B|A) = P(B) \quad (5)$$

that leads to the formal definition of independence:

$$P(A \cap B) = P(A)P(B) \quad (6)$$

which finally leads to the Bayes rule:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} \quad (7)$$

We will elaborate more on the Bayes rule when we describe the basics of Bayesian Networks and conditional probabilities.

2.2 Random variables

In some situations, outcomes of experiments are non-numerical, like tail or head when tossing a coin. However, statistical inference is usually expressed in numerical terms. A random variable is related to random outcomes typically associated with real numbers. We define a random variable (RV) as a function that maps (assigns) outcomes of experiments to a physical outcome (labels), typically real numbers. In the example of tossing a coin we could assign the value of 1 to the outcome ‘head’ and the value of 2 to the outcome ‘tail’.

A random variable typically assumes numerical values associated with each outcome from the original sample space. Random variables can be of two types, discrete and continuous, depending on a set of values taken. A discrete random variable takes only a very small (countable) number of values. For example, coin tossing outcomes are discrete. A continuous random variable may take any value in a given interval of possible values (not necessarily real numbers).

2.3 Conditional probability and random variables

Probabilities that are assigned to events can also be assigned to events in the random variable space. We define a set of random variables as: $X = \{X_1, X_2, \ldots, X_n\}$ from the domain that is being studied. A random variable can take a set of possible different values. Therefor a random variable in this context is a variable whose value is subject to variations due to change. The mathematical function describing the possible values of a random variable and their associated probabilities is known as a probability distribution. Similar to what we did define for events and conditional probability, we can do the same for random variables.
A conditional probability is the probability of a random variable occurring given the knowledge of another random variable. This is how we will define conditional probability for random variables. More formally, it can be expressed as follows:

$$P(X_i|X_j) = \frac{P(X_i \land X_j)}{P(X_j)}$$

(8)

Where $X_i$ and $X_j$ are random variables.

- $P(X_j)$ and $P(X_j)$ are the probabilities of $X_i$ and $X_j$ without regard to each other;
- $P(X_i|X_j)$ is a conditional probability. It is the probability of observing $X_i$ given $X_j$;

Combining equations such as:

$$P(X_i|X_j) = \frac{P(X_i \land X_j)}{P(X_j)} \quad \text{and} \quad P(X_j|X_i) = \frac{P(X_j \land X_i)}{P(X_i)}$$

(9)

Will lead to the Bayes theorem for random variables:

$$P(X_i | X_j) = \frac{P(X_j | X_i)P(X_i)}{P(X_j)}$$

(10)

The theorem is useful to swap $X_i$ and $X_j$ in conditional probability evaluation. The conditional probability $P(X_i | X_j)$ can be expressed by the conditional probability $P(X_j | X_i)$, $P(X_i)$ and $P(X_j)$. The theorem is often used based on pre-calculated probabilities, so called a-priori probabilities, which are based on prior research and used in probabilistic graphical models, discussed in the next paragraph.

### 2.4 Probabilistic graphical models

When representing or expressing conditional dependencies between random variables often so called probabilistic graphical models are being used. Different models represent different types of probabilistic dependencies. A probabilistic graphical model is a model for which a graph expresses the conditional dependence structure between random variables. The structure is used to show dependency relations and inference can use the structure to control the number of computations. Graphical models provide a basis for a number of efficient problems solutions such as: inference of visible, missing or hidden attributes and learning of model parameters and structure. Two branches of graphical representation are commonly used for this, namely directed models – e.g. Bayesian Networks and undirected models – e.g. Markov Networks. Both branches include the properties of factorization, and independences, but they differ in the set of independences they can encode and the factorization that they support. We will elaborate on them in the next paragraphs together with another modelling technique, called factor graphs (also used in FactorBase).
2.4.1 Bayesian Network

The network structure of a Bayesian Network (BN) is a directed acyclic graph. The model represents a factorization of the joint probability of all random variables (which are represented as the nodes in the model). Local semantics allows for computation of the likelihoods in an ordered, efficient fashion because the likelihood of any node is conditionally independent of its non-descendants given its parents. An example Bayesian Network model can be pictured as:

![Bayesian Network](image)

Probability theory has been used widely, and has been applied to a great variety of problems – e.g., life insurance, business decision making, gambling – but the problems have been constrained due to the computational infeasibility when dealing with large problems. The development of BNs together with the improvements of computer capacity allows more complex problems to be researched and solved.

As stated previously a BN is a graphical representation of the joint probability distributions for a set of random variables. A BN can therefore be used to build models for problems that deal with uncertainty. The nodes in a BN represent a set of random variables: $X = \{X_1, X_2, \ldots, X_n\}$. A set of direct links connects pairs of nodes, $X_i \rightarrow X_j$, representing the direct dependencies between the variables. The graphical representation is an acyclic direct graph with a conditional distribution for each node given its parents. More formally we denote this distribution by:

$$ P(X_i | \text{Parents}(X_i)) $$

(11)

When we dealing with complex BNs reducing the number of nodes in the calculations becomes very important, conditional independence helps to reduce the number of nodes. We will define conditional independence more formally: if $X_i$ independent of $X_j$ given $X_k$ we can write:

$$ P(X_i | X_k, X_j) = P(X_i | X_k) $$

(12)

As stated, in a BN the conditional independencies of the random variables in the BN are important. Suppose, $X_i$ represents ‘watering the garden’, $X_j$ represents ‘the weather forecast’ and $X_k$ represents ‘raining’. Initially, watering the garden is not independent of the
weather forecast; however, once we observe rain, they become independent. When we go back to our three variables $X_i, X_j$ and $X_k$ we can initially write:

$$P(X_i, X_j, X_k) = P(X_i)P(X_j|X_i)P(X_k|X_j, X_i)$$  \hspace{1cm} (13)

Assume $X_k$ is conditionally independent of $X_i$ given $X_j$ we can rewrite this to:

$$P(X_i, X_j, X_k) = P(X_i)P(X_j|X_i)P(X_k|X_j)$$  \hspace{1cm} (14)

reducing the number of conditions that need to be calculated.

Based on the previous described BN properties, we can state that a BN specifies the joint distribution in a structured form, with the following conditions:

- A node is a random variable;
- Each node has attached the numerical probabilities given its parents;
- The graph must be acyclic;
- The absence of an edge implies a conditional independence.

A BN reflects conditional independence statements. Namely that each variable is independent of its non-descendants in the graph given the state of its parents (Sucar 2015). With respect to the structure of the graph and the conditional independence relations we can formulate the following global semantics:

$$P(X_1, X_2, \ldots, X_n) = \prod_{i=1}^{n} P(X_i \mid \text{Parents}(X_i))$$  \hspace{1cm} (15)

where the left hand side is the full joint distribution and the right hand side is the factorization based on the structure of the graph (product of the local conditional distributions). This is explained in more detail in the upcoming example.
Consider the following example for the BN below with $X = \{A, B, C, D, E\}$ to explain (15):

![Bayesian Network with five nodes](image)

By specifying the conditional probabilities for all individual nodes we implicitly have also specified the global joint probability. For the directed graph in Figure 2.2 we can write:

$$P(A \land B \land C \land D \land E) = P(A)P(B|C)P(C|A)P(D|C, E)P(E|A, C)$$ (16)

A Conditional Probability Table (CPT) is a data structure that lists the probabilities of a variable given one or more other variables. Calculating the joint probability is a matter of multiplying the probability values stored in the CPT for the nodes involved. Below figure is an example whereby a BN is used to represent a model for a car that sends out a signal for help depending on the driver being in trouble or the car being broken. The signal can either be sent towards the emergency services or a pre-defined contact person.
A possible question for the model represented in the above figure could be: ‘What is the probability a signal was send to both the emergency services and the contact person, but the driver was not in trouble and the car was not broken?’ The full joint probability function for this example is according to (16):

\[
P(e \land p \land s \land \neg d \land \neg c) = \frac{P(e | s) P(p | s) P(s | \neg d, \neg c) P(\neg d) P(\neg c)}{P(e) P(p) P(s)}
\]

If we fill in the numbers we get: 0.9 * 0.7 * 0.001 * 0.999 * 0.998 = 0.00063

Another example, for the above BN we could also ask: ‘What is the probability of the driver being in trouble given the contact person was called?’ Again, according to (16) we can write:

\[
P(\neg e \land p \land s \land d \land \neg c) = \frac{P(\neg e | s) P(p | s) P(s | d, \neg c) P(d) P(\neg c)}{P(\neg e) P(p) P(s)}
\]

filling in the numbers we get: 0.10 * 0.70 * 0.94 * 0.001 * 0.998 = 0.000066

2.4.2 Markov Network

A Markov Network (MN) is a network structure represented using an undirected graph which can be cyclic. Thus, a Markov Network can represent certain dependencies a Bayesian Network cannot (and vice versa). Also for a Markov Network the set of random variables \( V \) are represented as nodes in the model together with a set of undirected edges, \( E \). Markov Networks also identifies local semantics for efficient inference, because the likelihood of a node is independent of any other node given its neighbors. An example Markov Network model can be pictured as:

![Figure 2.4 A Markov Network with five nodes](image)

Formally, a Markov Network is set set of random variables, \( X = \{X_1, X_2, \ldots, X_n\} \) that are indexed by \( V \), such that \( G = (V, E) \) is an undirected graph, which satisfies the Markov property: a variable \( X_i \) is independent of all other variables given its neighbors \( \text{Neighbor}(X_i) \):

\[
P(X_i | X_1, \ldots, X_{i-1}, X_{i+1}, \ldots, X_n) = P(X_i | \text{Neighbor}(X_i))
\] (17)
Exploiting relational database technology for statistical machine learning in FactorBase

The neighbors of a variable are all the variables that are directly connected to it in the graph. In the previous paragraph we described an example showing the calculation of the joint probability for Bayesian Networks. In the below example we elaborate on the joint probability using a Markov Network.

![Markov Network with six nodes](image)

The concept of a clique in a Markov Network is important. We define a clique as a complete sub-graph of $G$. In the above example graph we can distinguish three cliques: $C_1 = \{A, B, C\}$, $C_2 = \{C, D, E\}$ and $C_3 = \{D, E, F\}$. A Markov Network is specified numerically by associating potentials with the cliques of the graph. A potential is defined as the function on the set of configurations of a clique that associates a positive real number with each configuration. Hence, for every subset of nodes $C_i$ that forms a clique, we have an associated potential $\emptyset_i(C_i)$. The joint probability for all nodes in the graph is obtained by taking the product of the clique potentials:

$$P(X) = \left(\frac{1}{Z}\right) \prod_{C \in \text{cliques}(G)} \emptyset_c(X_C) \quad (18)$$

where $Z$ is the normalizing constant and $\emptyset_c$ is a local function over the variables in the corresponding clique $C$. Some other important properties of Markov Networks can be pictured as:

![Markov Network properties](image)
Exploiting relational database technology for statistical machine learning in FactorBase

\[
P(A, B, C) = \frac{\phi_{AC}(A,C)\phi_{BC}(B,C)}{k}
\]

(19)

\( A \) and \( B \) may be unconditionally dependent: \( P(A, B) \neq P(A)P(B) \)  (20)

\( A \) and \( B \) are conditionally independent on \( C \): \( P(A, B|C) = P(A|C)P(B|C) \)  (21)

(2) Marginalizing over \( C \) makes \( A \) and \( B \) dependent

(3) Conditioning on \( C \) makes \( A \) and \( B \) independent.

Both Bayesian Networks and Markov Network models are used to model graphical probabilistic models. In the next paragraph we describe another modeling technique called factor graphs. This modeling technique is often used for inference (see paragraph 2.4.6) because there are very efficient inference algorithms available using this modeling technique.

2.4.3 Factor graphs

Both Bayesian Networks and Markov Networks can be represented using factor graphs (Kschischang 2001). Factor graphs explicitly express the factorization structure of the corresponding probability distribution. A factor graph consists of a tuple \( (X,F) \), where \( X \) is a set of RVs and \( F \) is a set of factors. Each factor is a function from the values of \( X \) to the non-negative real numbers. Factor graphs are often used to represent two of the most common types of graphical models. Bayesian Networks and Markov Networks. Both models use a graphical representation whereby RVs are used as nodes and implicitly provide the factors through the graph structure. Below we picture an example Bayesian network structure with the corresponding factor graph (factors represented as black squares) and conditional probability table (CPT) (Kimmig 2015):

![Factor graph and Bayesian Network](image)

Figure 2.7 A factor graph with corresponding Bayesian Network

The figure above represents the Bayesian network structure (a) and the corresponding factor graph (b). The vertices are the RVs in the network and the probability distribution is specified by providing the conditional probability for each node given the value of its parents. The simplest way of expressing the conditional probabilities is via CPTs (c).

As previously explained Markov network is an undirected graphical model where the nodes correspond to the RVs in \( X \). The model is used to compute the probability distribution
Exploiting relational database technology for statistical machine learning in FactorBase

over \( X \) as a product of potential functions defined over the cliques in the graph. A clique in an undirected graph \( G = (V, E) \) is defined as a subset of the vertex set \( C \subseteq V \), such that for the vertices in \( C \), there exists an edge connecting the two. In an example picture using a factor graph we can represent such network as (Kimmig 2015):

In the above example the Markov network (a) with the factor graph (b) (factors represented as black square box), has two cliques (ABD and BCD). Therefore there are two potential functions in the factor graph (c), one over the variables A, B and D and one over the variables B,C and D. In the example the RVs are Booleans.

2.4.4 Markov blanket

An important concept in learning graphical models is the Markov Blanket. We define a Markov Blanket of node \( n \) as the union of \( n \)'s parents, \( n \)'s children and the parents of \( n \)'s children. 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. An example Markov Blanket in a Bayesian Network can be pictured as:

A Markov blanket in a graphical probabilistic network is easier to represent and analyze than in a Bayesian Network since it does not include any nodes at a distance larger than one. In a Bayesian Network however there can be a relation through parents of children and thus a remaining dependence is much more difficult to evaluate.
2.4.5 Bayesian Networks versus Markov Networks

Probabilistic graphical models in general, like Bayesian Networks and undirected Markov Networks are used to compactly represent joint probability distributions over a set of random variables. The main purpose of this models is probabilistic inference (e.g. queries like “what is the posterior distribution of the random variable Salary, if observed Age and education”).

With a Markov Network we can represent cyclical dependency relations, with Bayesian Networks we can represent induced dependencies. Depending on the needs and the data to analyze it sometimes beneficial to convert a Bayesian Network to a Markov Network (or vice versa) before using the model for analysis.

While converting a Bayesian Network to a Markov Network (also done in FactorBase), we have to make sure that the conversion of a directed model (Bayesian Network) to an undirected model (Markov Network) does not result in any loss of the dependencies in the network. All dependencies in the Bayesian Network are potential dependencies in the Markov Network and the structure must be able to handle any evidence. The conversion method used is often based on a moralization approach (Khosravi 2012), where common parents of a node have to be connected to represent the additional dependencies. We can picture such a conversion as:

![Conversion example from Bayesian to Markov Network]

*Figure 2.10 Example conversion of a Bayesian Network to a Markov Network*

By converting the Bayesian Network model from a directed model to an undirected model we avoid the problem with cyclic dependencies. At the same time the concept of a Markov Blanket as explained previously is used for optimization when determining the inference in the Markov Network. In the next paragraph we discuss inference in more detail.

2.4.6 Inference

Probabilistic inference consists of propagating the effects of certain evidence in a network model to estimate the effect on the unknown variables (Sucar 2015). By knowing the values

---

3 Inference in this context is defined as deriving rational conclusions from premises known or assumed to be true.

4 The most widely used method for inference in a Markov Network is Markov Chain Monte Carlo, and in particular Gibbs sampling, and when using this method each variable is sampled given its Markov Blanket.
(of a subset) of the variables in the model, the posterior probabilities of the other values are obtained. If the (sub)set of known variables is empty, the idea is to obtain the posterior probabilities of all the variables in the model. In short: the main goal of inference is estimate the values of hidden nodes (attributes) given the values of visible nodes (attributes).

Basically there are two variants of the inference problem in graphical models: ‘computing marginals’ and ‘most probable explanation’ (MPE) inference (Kimmig 2015).

When computing marginals we are interested in the posterior probability of a single variable, $H$, given a subset of known variables, $E$ (that is $P(H|E)$). Specifically, we are interested in the marginal probabilities of the unknown variables in the model. We will refer to this as single query inference (Sucar 2015). We can also compute the marginals for a set of variables. In that case we are interested in calculating the posterior probability of a set of variables $H$, given the evidence $E$ (that is $P(H|E)$). It can be solved by computing marginals several times by applying the chain rule. For example, $P(A, B | E)$ can be written as $P(A | E)P(B | A, E)$ which requires two times a single query reference followed by a multiplication.

For MPE it is of interest to know which are the most probable values in a set of hypothesis. This can be formulated as: $\text{ArgMax}_H P(H|E)$, whereby $H$ includes all non-observed variables. This is also known as the total abduction problem. When we are interested in the most likely joint state of some of the non-observed variables it corresponds to the maximum a posteriori assignment (MAP) or partial abduction problem (Sucar 2015).

Solving the complexity of the above tasks is exponential in the worst case. In practice however they can be solved efficiently. Next we summarize two common inference techniques for graphical models.

### 2.4.6.1 Variable elimination

The variable elimination technique is based on the idea of calculating the probability by marginalizing the joint distribution. It takes advantage of the independence conditions of the network, together with the distributive properties of addition and multiplication (Sucar 2015). Assume we have a network with the joint probability distribution of $X = \{X_1, X_2, ... X_n\}$. We want to calculate the posterior probability of a (sub)set of variables, $X_H$, given a subset of variables, $X_E$. The remaining variables are $X_R$, such that $X = X_H \cup X_E \cup X_R$. We can formulate the posterior probability of $X_H$ given the following evidence as:

$$P(X_H | X_E) = \frac{P(X_H, X_E)}{P(X_E)} \quad (22)$$

We can now obtain both terms via marginalization of the joint distribution:

$$P(X_H, X_E) = \sum_{X_R} P(X) \quad (23)$$

and

$$P(X_E) = \sum_{X_H} P(X_H, X_E) \quad (24)$$

The objective of the technique is to perform these calculations as efficiently as possible. To achieve this, the joint distribution is represented as a product of the local probabilities according to the network structure. Then summation is executed only on the subset of terms which are a function of the variables being normalized.
An efficient mechanism for inference can be viewed as passing a message (information) to a neighboring vertex in the graph. We can calculate a marginal of any singly-connected graph by starting at a leaf of the tree, eliminating the variable there, and then working inwards, removing each time a leaf of the remaining tree. Provided we perform elimination from the leaves inwards, then the structure of the remaining graph is simply a sub-tree of the original tree, with the CPT entries modified under recursion. This is guaranteed to enable us to calculate any marginal using a number of summations which scales linearly with the number of variables in the graph (Barber 2012). Consider the following example network:

![Figure 2.11 Example of directed network with five nodes](image)

As an example, we want to obtain $P(A \mid D)$. For this to work, we first need to obtain $P(A, D)$ and $P(D)$. To calculate the first term, we have to eliminate B, C and E from the joint distribution. This can be written as follows:

$$P(A, D) = \sum_B \sum_C \sum_E P(A) P(B \mid A) P(C \mid A) P(D \mid B, C) P(E \mid C)$$  (25)

If we distribute the summation, we can also write:

$$P(A, D) = P(A) \sum_B P(B \mid A) \sum_C P(C \mid A) P(D \mid B, C)$$  (26)

If we assume that all variables are binary the above logic implies a reduction from 32 operations to 9 operations. The reduction is even greater when we would use a model where a variable can have more than two values. Belief propagation is such an efficient algorithm for inference, we will elaborate on it with a more detailed example in the next paragraph.

2.4.6.2 Belief propagation
Belief propagation is an efficient algorithm for computing marginals in a graphical network. We refer to an algorithm as proposed by Pearl (Pearl 1998). The main idea behind belief propagation is that each node in the graph sends ‘messages’ to all of its neighbors, based on the ‘messages’ received from the other neighbors. It calculates the marginal distribution for unobserved nodes, conditional to any observed nodes.
Belief propagation is about sending messages. To be able to send messages between nodes a compatibility matrix $\psi$ is used. This matrix represents the statistical dependencies between nodes. For example, in a medical diagnosis system, one node might represent a symptom, and the nodes that are linked to are related symptoms or diseases. The numerical value of the compatibility matrix represents the strength of the relation between nodes. To send a message we can state: multiply together all in the incoming messages, except the nodes you are sending to, then multiply by the compatibility matrix and marginalize over the sender’s states. This can be formulated as:

$$M_i^j(x_i) = \sum_{x_j} \psi_{ij}(x_i, x_j) \prod_{k \in \text{Neighbor}(j) \setminus i} M_j^k(x_j)$$

To illustrate this, we use a simple belief network based on the previous graph example in below figure, using the following factor graph with some details for the features F1 and F2:
Now assume that we want to calculate the joint probabilities of A, B and C without using belief propagation. We first have to calculate the joint distribution and then sum out the variables. The joint distribution can be found by multiplying the features F1 and F2. If we do so, we acquire the joint distribution as follows:

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th>Pot.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>8</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>16</td>
</tr>
</tbody>
</table>

We can calculate that \( P(A=0) = 14/42 = 1/3 \) (\(2+2+2+8 / 2+2+2+8+4+4+4+16\)), \( P(B=0) = 12/42 = 2/7 \) and \( P(C=0) = 12/42 = 2/7 \). Now we can do the same using belief propagation. In that case each node will send a message to its neighbor node. A message is the belief about the node’s probability of being in either 0 or 1 state. We can represent this in a table of two entries specifying the un-normalized probability of a node. We can represent a message as a row factor. From our factor graph we could write \( \Phi(A,B) \) as \([1 2 2 4]\). A node sends a message to a feature F by multiplying all messages it gets from its neighbor nodes except F. A factor F sends messages to a node n by multiplying all the messages F gets from its neighbor nodes except for n, followed by multiplying the result product with its own potential table and finally summing out all variables except n (Pearl 1998).

All nodes start the process in parallel. Since they did not receive any message from the factors, message \([1,1]\) is send towards all factors. Now we calculate the messages from F1 to A, we multiply all messages to F1 except from A. There is only one message from B. We then multiply this message from B with the F1’s table to get \([1 1 2 4]\). After summing out B we get the message from F1 to A as \([3 6]\) (\([1 3] + [2 3]\)). Similarly, this is done for message from F1 to B \([1 2]\), F2 to B \([4 5]\) and F2 to C \([1 2]\).

Next step is to calculate messages from B to F1 and B to F2. All messages from B to F1 is multiplication of all messages to B except for F1 to B. Since there is one remaining message to B i.e. from F2 to B, we do not need to multiply anything and simply send \([4 5]\) from B to F1. Next to this B sends \([1 2]\) to F2.

Again, factors send messages to nodes. To calculate a message from F1 to A, we multiply F1’s table with messages from B to F1, resulting in \([4 10 8 20]\), and after summing out we get \([14 28]\), which is \([1 2]\). Similar we do the messages from F1 to B \([1 2]\), F2 to B \([4 5]\) and from F2 to C \([4 10] = [2 5]\).
Nodes now again send messages to factors. Message from B to F1 is [4 5] and from B to F2 is [1 2]. Factors send messages to nodes, message from F1 to A is the same as the previous [1 2]. Similarly, F1 to B is same as the previous [1 2], from F2 to B is also the same [4 5], and F2 to C is also the same [2 5]. From here the algorithm is stopped because the same messages are send from the factors to the nodes.

To calculate the marginal of each node we need to multiply all messages to that node. This results in: \( \Phi(A) = [1 2] \), \( \Phi(B) = [1 2] \times [4 5] = [4 10] = [2 5] \), \( \Phi(C) = [2 5] \). Having the calculated messages, we can the associated probabilities: 
\[ P(A=0) = 1/3, \quad P(B=0) = 2/7 \quad \text{and} \quad P(C=0) = 2/7, \]
which is the same as calculated earlier using the joint distribution.

2.4.7 Learning

Learning a graphical model consists basically of parameter learning and structure learning. Parameter learning refers to learning of the conditional probabilities given the structure of the network (i.e. knowledge of interdependencies). Structure learning refers to learning the connectivity within the network (i.e. the interdependencies). Below we elaborate on both aspects.

If you derive predictions from a model it is required that the values for the parameters are estimated. For this we need to learn parameters from the data. We can picture this as:

Maximizing the data likelihood is the basic parameter estimation method for Bayesian Networks. The maximum likelihood estimates equal the observed frequency of a child value given its parent values. This is also the approach used in FactorBase.
The goal of structure learning is given the data to discover the dependency structure of the model and the parameters of the potential functions. We can picture this as:

A naïve approach to learn the structure could be to enumerate all possible network structures and choose the one that maximizes some pre-defined criteria. The problem with this approach is that it becomes unfeasible when the number of nodes increase (for example 10 nodes will lead to $O(10^{18})$ structures). Most approaches are based on a search and score approach. The idea behind this is to have a score function for measuring model quality (like a penalized likelihood) together with a search function to find a maximum score. This function assigns a score $S(G|D)$ to the graph. The goal is then to find the best $S(G|D)$ given dataset $D$.

2.5 Multi-relational learning
In this paragraph we summarize the key characteristics of multi-relational learning by providing background information on single relation learning versus multi-relational learning, an example to illustrate the difference between multi-relation and single relational learning followed by a description of multi-relational learning tools.

2.5.1 Background
Many real-world applications store their data in relational databases. A relational database typically consists of several tables (relations) and not just one table. Machine learning techniques are traditionally applied to data stored in a single table, in this table the data is stored in a non-relational or flat file format (Mitchell 1997). Each row in this table corresponds to an instance and each column corresponds to an attribute. The language used to access the data is typically based on propositional logic, where the propositions are of the form: “attribute $\otimes$ value”, where $\otimes$ is an element of a pre-defined set of operators such as $>$, $\geq$, $<$, $\leq$, $=$ (Lavrac 1994). The goal of single table learning is often to represent predictive dependencies between the attributes of the rows stored in a single table. Multi-relational learning is required in domains where the data is highly structured and stored in multiple tables in a relational database (Dzeroski 2001). The goal for multi-relational learning is to
represent dependencies between attributes of rows that are related or linked to each other using multiple relations (or tables). The field of Statistical Relational Learning (SRL) aims to extend the machine learning algorithms to relational data that consists of multiple relations (tables) (L. T. Getoor 2007). To do this effectively the SRL needs to provide two important aspects: (i) it needs to provide a language for expressing dependencies between different types of entities (tables), the attributes and the relation between the attributes; and (ii) it needs to allow for probabilistic reasoning in a relational environment (Kimmig 2015, page 4).  

2.5.2 Example

To illustrate the concept of multi-relation learning versus single relation learning we use an example:

\begin{figure}[h]
\centering
\includegraphics[width=0.5\textwidth]{example.png}
\caption{A database representation of three tables storing accidents}
\end{figure}

In the above picture we see three tables that are used store information on car accidents in a relational database. Assume we want to find information on accidents using a query such as:

"If an accident takes place in a road with maximum speed of 100km/h, and involves a car whose driver is not wearing a seat-belt, then the accident is likely to be fatal?"

We can see that querying the data will involve all three relations. Relational patterns are typically expressed in subsets of first-order logic (also called predicate or relational logic). We can rephrase the above query more formal into:

\[
\begin{align*}
\text{Driver(Age, Gender, Nationality, No)} & \leftarrow \\
\text{Car(Type, License)} & \land \\
\text{Accident(Date, >100km/h, Yes)}
\end{align*}
\]

This pattern says, “A driver is likely to have a fatal accident in his car if he is not wearing a seatbelt and drives more than a 100km per hour”. The above approach is referred to as first order learning approach or relational learning approach; as the pattern it finds is expressed in the relational formalism of first order logic.
Multi-relational learning algorithms are used to find patterns valid in a given database. Many of these algorithms come from the field of inductive logic programming (ILP). Situated at the intersection of Machine Learning and logic programming, ILP has been concerned with finding patterns expressed as logic programs. In recent years, however, the scope of ILP has broadened to cover the whole spectrum of multi-relational learning tasks (classification, regression, clustering, association analysis). The most common types of patterns have been extended to their relational versions (relational classification rules, relational regression trees, relational association rules).

Multi-relational learning algorithms tightly integrate with relational databases. Because relational databases can be huge, most of these algorithms pay much attention to efficiency and scalability, and use techniques such as sampling and pre-computation (or materialized database views).

2.5.3 Multi-relational learning tools

The intersection of database -management and machine learning using Bayesian Networks is heavily researched at this moment. Database researchers did notice the usefulness of statistical relational models for knowledge discovery and representing uncertainty in databases. However, often these models focus on inference given a statistical relational model, not on learning the model itself from the data stored. Most of these solutions for learning from a given statistical relational model in a database are based upon the following design:

![Figure 2.17 A generic design for a SRL solution for learning multi-relational data](image)

A different approach is FactorBase, an SQL based framework to leverage database management systems to support multi-relational model discovery (i.e. learning a model). With FactorBase the graphical representation of the model is pushed inside the database allowing the researchers to use SQL as a high-level scripting language for Statistical Relational Learning (SRL) based on multi-relational data. Experiments show significant
Exploiting relational database technology for statistical machine learning in FactorBase

scalability gains with respect to the number of records researched and the ability to discover more complex cross-table relations than previous SRL based systems such as BayesStore (Wang 2008) and Tuffy (Niu 2011).

In paragraph 2.6 we present a high-level overview of FactorBase. It uses the concept of Bayesian Networks, Markov Networks and Probabilistic Relational Models (L. Getoor 2001). Probabilistic Relational Models are an extension of Bayesian Networks whereby the nodes of the network are attributes (columns) of relations (tables) in the database. The edges in such networks represent probabilistic dependencies between the attributes. A dependency is either TRUE or FALSE, which allows propositional logic to be used when querying these dependencies. The semantics of the Probabilistic Network is a probability distribution over the attributes and the relations between these attributes (L. Getoor 2001). A probabilistic relational network consists of two components:

• The parameters (stored attributes of all relations). The parameters represent the conditional probability distribution and;
• The structure of the network itself.

To learn a probabilistic relational network one first needs to examine the attributes and store meta-data about the data. Key here is the likelihood function, or the probability of the data given the model. In general, one can state that the higher the probability, the better the model to predict the data. To learn the structure of a network, three components need to be defined (L. Getoor 2001):

• The hypothesis space or stated differently: the number of structures the algorithm can consider;
• A scoring function that evaluates each hypothesis found;
• The search algorithm used. Most of the algorithms used are based on the category of ‘hill-climbing search’ algorithms\(^5\).

2.6 FactorBase, a high-level overview

In this paragraph we describe a high-level overview of FactorBase (Olivier Schulte 2015). As mentioned before, there are several systems that leverage relational database support for learning. Often, the data is represented in a single table or data matrix. FactorBase uses a mechanism to support graphical model learning for multi-relational data stored in multiple tables (or relations) in a relational database.

Learning Bayesian Networks includes two aspects: learning the structure and learning the parameters (see 2.4.7). One the most popular methods of learning a Bayesian Network structure from data, is the score-based approach. The process assigns a score to each candidate Bayesian Network, typically one that measures how well that Bayesian Network describes the passed in data set. When the structure is known, parameter learning consists of estimating the conditional probability tables (CPT) from the data. If we have sufficient data for all variables, learning is straightforward (for now we assume this is the case). The CPT for each variable can be estimated from the data based on the frequency of each value (or combination of values) obtaining a maximum likelihood (ML) estimator of the

\(^5\) In a hill-climbing heuristic, one starts with an initial solution. Then you generate one or more neighboring solutions, pick the best solution and continue until there are no better neighboring solutions. This will generally yield one solution. In hill-climbing, we need to know how to evaluate a solution, and how to generate a "neighbor".
Exploiting relational database technology for statistical machine learning in FactorBase

parameters (see for an example later on). Picture-wise we can describe the FactorBase system-flow as:

![FactorBase system flow](image)

Figure 2.18 The overall system flow in FactorBase with the three components

Based on a given database with data, it constructs a meta-data model exploring the database system catalogs, which contains a description of the tables and data stored. This is what is called the ‘Variable Manager’ component.

Using the second component called the ‘Count Manager’, it then builds for each node in the graphical representation a CPT and a Contingency Table (CT). The CPT contains the maximum likelihood estimates for given variable and its values, the CT contains the data value counts for a given variable. In FactorBase, CPTs are created for each column attribute and relation between tables, based on the data model stored. A CT in FactorBase is a table that has a row for each possible attribute value and a count that records the count of the corresponding query, which is used to construct the CPT.

Another aspect of learning is structure learning. Structure learning is about obtaining the topology of the BN based on the data stored. For structure learning there are two main types of methods: (i) global methods based on search and score and (ii) local methods that use conditional independence tests (Sucar 2015). FactorBase is using a method based on search and score. This component is called the ‘Model Manager’.

In the example presented next, we will show how the various components are used and generate the previously described output. We will picture an example with three relational tables representing an ordering system for books. First an example of data stored in these tables:
Exploiting relational database technology for statistical machine learning in FactorBase

An example of how a relational Bayesian Network model for the OrderedBooks relation from Figure 2.19 looks is shown in the figure below:

In the above figure we see the BN for the OrderedBooks (OB) relation. The columns from other tables such as Priority from the relation Orders is linked via a node Priority(O), another column Readability from the relation Books is linked via the node Readability(B). Important to note for this figure is that the nodes refer to a set of random variables for a specific column and not to a single random variable.
Exploiting relational database technology for statistical machine learning in FactorBase

This model can be built querying the database system catalogs that store the information on tables and table relations (a so-called Schema Analyzer). When we have this model represented in a BN we can easily generate the SQL for it using the SQL framework described by Knobbe (Knobbe 1999).

The next step is to build CPTs and CTs for each node using SQL. Because the structure of the data-model is known, the SQL can be generated. In the picture below we see two of these tables (with fabricated data) for the Discount(O,B) node and its parent OrderedBooks(O,B) pictured in Figure 2.20. The data is generated by querying the tables that store the actual Order and OrderedBook data.

<table>
<thead>
<tr>
<th>Discount(O,B)</th>
<th>OrderedBooks(O,B)</th>
<th>CP</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>T</td>
<td>0.45</td>
</tr>
<tr>
<td>10</td>
<td>T</td>
<td>0.25</td>
</tr>
<tr>
<td>5</td>
<td>T</td>
<td>0.10</td>
</tr>
<tr>
<td>0</td>
<td>T</td>
<td>0.20</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Count</th>
<th>Discount(O,B)</th>
<th>OrderedBooks(O,B)</th>
</tr>
</thead>
<tbody>
<tr>
<td>90</td>
<td>2</td>
<td>T</td>
</tr>
<tr>
<td>50</td>
<td>10</td>
<td>T</td>
</tr>
<tr>
<td>20</td>
<td>5</td>
<td>T</td>
</tr>
<tr>
<td>40</td>
<td>0</td>
<td>T</td>
</tr>
</tbody>
</table>

*Figure 2.21 CPT and CT example for a column from the table OrderedBooks*

In the CPT in Figure 2.21 we see the frequencies of values for the column Discount. All values are displayed and the CP column indicates the frequencies using percentages, i.e. how often that value is stored. In the CT we see the total counts for each Discount value. The column OrderedBooks(O,B) indicates with the value T that there is a relation between Discount(O,B) and OrderedBooks(O,B), as we did picture in Figure 2.20. All the generated CPTs and CTs are stored in the database.
Exploiting relational database technology for statistical machine learning in FactorBase

Both the CPTs and CTs are generated executing SQL. The generic SQL used to create such tables will be of the following form:

```
CREATE TABLE <CT> (variable list) AS
SELECT COUNT(*) AS count, (variable list)
FROM (table list)
GROUP BY (variable list)
WHERE (join conditions)
```

*Figure 2.22 The SQL for the creation of a CT for a list of variables*

for the CT. Often, instead of creating a table inside the database, a so-called view is created using the CREATE VIEW syntax, replacing the CREATE TABLE syntax. A view is a table, which is built on the fly when queried. The above query is also used to store statistics on the relationships between attributes. Negative relationships cannot be determined using the above query, but are taken care of in a different way (Zhensong Qian 2014).

For the CPT the generic SQL query will be of the following form:

```
SELECT count/temp.parent_count AS CP,
(variable-list)
FROM (CT)
NATURAL JOIN
(SELECT sum(count) AS parent_count,
(variable-list)
FROM (CT)
GROUP BY (variable-list)) AS temp
```

*Figure 2.23 The SQL used to determine the CPT for a variable*

The results are then stored in the relational database and used as input for learning the structure and parameters.

In FactorBase a model manager based on search and score is used to learn the model structure (see paragraph 2.4.7) and store the result in a model database (graphical model and parameter values). It also supports the construction and querying of large structured statistical models (O. K. Schulte 2012).

The next step is to construct models using the ‘Model Manager’ Component and store them in the model database. This construction consists of the following steps (Olivier Schulte 2015, page 5-6)

- Compute parameter estimates for the model using the statistics from the generated CTs using SQL SELECT statements with natural joins;
- Compute model characteristics such as number of parameters or degrees of freedom in the model using SQL;
- Compute a model selection score that quantifies how the model fits the relational data stored in the database.
Score-based methods search for the model structure that best matches the data by introducing a scoring function that evaluates each model candidate with respect to the passed in data.

Running FactorBase against a data-set consists of several steps. The database schema is created and the data is loaded using SQL scripts (O. Schulte 2015). Various configuration files need to be updated with user login credentials. To run the actual test a terminal session is used to start the FactorBase java program (see appendix 11.1 for an example).

2.7 Relational databases
In this paragraph we describe some of the relational database concepts and techniques we will use later on when describing our new framework for using dynamic views for multi-relational learning. We first introduce the concept of views in a relational database followed by a description of prepared statements and stored procedures.

2.7.1 Views in a relational database
We will define a view as the result set of a stored query on data in a relational database (Date 1990, page 11-12). Unlike an ordinary (base) table a view is not part of the physical database schema. It is a result set and with that a virtual table dynamically constructed by the database engine when the stored query is executed. The stored query essentially consists of a relational retrieval expression called a SELECT. An example of such a SELECT used for a view could be:

```sql
CREATE VIEW CA_VIEW AS
    SELECT id, first_name, last_name
    FROM employees
    WHERE zipcode in (SELECT zipcode
                      FROM locations
                      WHERE state = "CA")
```

*Figure 2.24 SQL example for creation of a database view*

In the above example a SELECT statement on the view CA_VIEW will generate a result set with rows from the base table employees whereby all employees are shown that live in the state ‘CA’. In most database engine views can also refer to other views in the FROM clause (i.e. they can be nested).
Exploiting relational database technology for statistical machine learning in FactorBase

We can picture the anatomy of a view as:

![Figure 2.25 The anatomy of a database view pictured](image)

Views can provide advantages over ordinary tables. A view can represent a subset of the data contained in the underlying table(s). Views can also join and simplify multiple tables into a single virtual table. Also a view can act as an aggregated virtual table, the database-engine aggregates data (using sum, average etc.) and represents the data as part of the virtual table. Views can be used to hide sensitive information, thus providing a form of data security. Finally, one of the major advantages is that a view takes little space, the database only contains the view definition and not a copy of the actual data the view represents.

2.7.2 Prepared statements
To generate and execute a SQL statement dynamically the database-engine must support a concept called prepared- or parameterized-statements. Each database engine can have its own specific implementation but in general such a statement takes the form of a template into which certain constant values are substituted during execution. For MySQL to use a prepared statement you need to use three other MySQL statements as follows:

![Figure 2.26 MySQL statements needed to use prepared statements](image)

‘Prepare’ to prepare the statement for execution, ‘execute’ to execute the prepared statement and ‘deallocation prepare’ to indicate that the associated resources can be released.
Exploiting relational database technology for statistical machine learning in FactorBase

An example of a prepared statement could be:

```
SELECT *
FROM PRODUCTS
WHERE productName = ?
```

*Figure 2.27 SQL query example for a prepared statement*

When the database engine executes the above query with different values for the column ‘productName’, the database engine does not have to parse the query every time, this will speed up the execution of the statement increasing the performance because the result set is returned faster. The above prepared statement functionality can also be used to execute SQL strings. For example, we store the above statement in a database string variable, execute the statement referring to the variable and then release the associated resources.

The above example could be re-written to:

```
SET @SQLQUERY = ‘SELECT *
FROM PRODUCTS
WHERE productName = ?’
PREPARE stmt from @SQLQUERY
SET @productname = ‘CAR’
EXECUTE stmt USING @productname
DEALLOCATE PREPARE stmt
```

*Figure 2.28 Example SQL query for a prepared statement using a SQL string*

From the above it is clear that we can generate any SQL statement, store it in a SQL string variable and then execute it using the prepared statement feature. This concept will be used when generating dynamic views for multi-relational learning (see chapter 5). The prepared statement feature can be used without using the so called placeholders in the SQL query we execute.
2.7.3 Stored procedures

To execute the SQL that implements the dynamic views for multi-relational learning (see chapter 5.5.1) we use stored procedures. A stored procedure is set of SQL statements stored in the database. Instead of executing each individual statement, a SQL client can refer to the stored procedure (MySQL, Using stored routines (procedures and functions) 2017). An example of a stored procedure could be:

```sql
DROP PROC p1;
CREATE PROC p1(id int)
BEGIN
    SELECT Name from employees
    WHERE employee_id = id
END;
```

*Figure 2.29 Example SQL syntax to create a stored procedure*

The client program from that point can execute the code by simply referring to the stored procedure:

```sql
EXEC P1 <id>;
```

*Figure 2.30 Example SQL to execute a stored procedure*

The result is returned to the client executing the stored procedure. The main reason for using stored procedures is that the code is stored in the database and we can develop code using functions/procedures which are better to maintain than one single batch of SQL statements. Another advantage is that SQL stored in the database is already parsed and compiled in the database. This reduces the interaction between the client program and database with a positive effect on performance.
3. Research design

There are various opportunities to improve the relational database operations and algorithms used in FactorBase (Olivier Schulte 2015, Future work, page 10). In this paragraph we will elaborate on this in more detail in preparation of the research questions we describe later on. First we picture the research model we want to use as:

![Research model used](image)

The model is used to scope the research and add direction conducting the various experiments.

The multi-relational algorithms and theory and the SQL and relational database logic for multi-relational learning necessary for the research are described in chapter 2. The relational database mechanisms for data storage are described in chapter 2.7.1.

Input for experiments of the research are the datasets used in FactorBase (Olivier Schulte 2015, page 7), which are six real-world datasets. The datasets, source code used, information on the algorithm used, models and information on the system used are publically available (O. Schulte 2015). The research will start with a setup similar to the original research, so a baseline can be created using the six datasets. From here the research will start on the various improvements. Each improvement will be tested using the six datasets and is compared against the established baseline. The research will focus on improvements in the following areas:

- Reduction in time needed to learn the data and structure from the datasets used. In the research questions we will refer to this as performance;
• Reduction in resources necessary such as memory and disk space needed to learn from the datasets used. In the research questions we will refer to this as resource usage.

The opportunities for improvement could be in the area of algorithms used to generate the CPTs and CTs from the data stored. Also the SQL used to implement these algorithms has room for improvement. On the other hand, leveraging specific relational database features such as view materialization, column versus row storage and in-memory processing are areas that can be explored to further improve the relational database operations.

3.1 Research questions
The research questions are derived from the following research objective:

“To which extent can we further exploit relational database technology for statistical machine learning in FactorBase?”

This research objective is accomplished by getting an answer on the following research questions:
• RQ1: What improvements can be made in the algorithms used to build the CPTs and CTs in FactorBase that result in a better performance for generating the associated CPTs and CTs for the input datasets?
• RQ2: What improvements can be made for the algorithm used (Zhensong Qian 2014) for determining the negative relationships, i.e. the lack of a relationship in the datasets, when creating the CTs for these datasets?
• RQ3: What improvement can be made for the SQL algorithms in FactorBase (Olivier Schulte 2015, Appendix) with respect to the performance and resource usage when processing the datasets?
• RQ4: What is the effect on the performance and resource usage when using a column-store for the CPT and CT statistics versus using a typical row-store mechanism in the available RDBMS?

3.2 Research scope
The scope of the research is limited with respect to the number of hours that one can spent on the research. To limit the research, we will:
• Only focus on the specific algorithms used for parameter learning and model structure learning in FactorBase. These algorithms are used to generate SQL which is then used to read the data from datasets and generate the CPTs and CTs for these datasets;
• Limit our selves to relational database systems, which are publically available (MySQL for example);
• Use the published datasets used in the FactorBase research (O. Schulte 2015). A baseline is created and from there enhancements are described and the results are published compared to the baseline.
3.3 Research validation

The main question for the validation of the results is: ‘have the proposed enhancements a positive effect on the time needed to discover a multi-relational model using FactorBase?’.

To validate this question we first create a baseline using the published data sets (O. Schulte 2015). The baseline is executed for six real world datasets. FactorBase generates output that divides a run for a data set in five steps taking a certain amount of time:

- Generic building of environment (from here building time):
- CT building time;
- Smoothed CP time;
- CSV pre-computer;
- Parameter learning;
- Structure learning.

The time needed for each step is measured and a graph for data set is created to reflect the time spent (see 7.1). At the same time each FactorBase run will generate output which consists of:

- The CT and CPT tables generated in the database;
- The CT and CPT data available as .csv files in a data set output directory;
- The learned models available in SQL table/data format and BIF/XML files.

The output of the baseline will be saved for each data set.

The focus in this research will be on the FactorBase steps: CT building time and Parameter Learning time. These two steps will take most of the time of all steps executed. The new proposed framework will generate CT and CPT tables too. The content of these tables then is validated by comparing the content with the contents of the .csv files generated by FactorBase and the CPT and CT tables stored in the database after each FactorBase run. When the correctness of the new algorithms is confirmed, the new logic is executed a number of times and the time measured for each run is used to generate a graph for each data set. For each data set a final graph is generated with the baseline times and the times needed for the new algorithms to execute the same amount of work.

To measure the execution time and resources needed using the new proposed framework we will use the MySQL performance monitor tools. These tools are by default available and track all aspects of database performance (see 7.2.1 for more details).
4. FactorBase: Learning multi-relational graphical model example

To be able to show the contribution of dynamic views we will first elaborate on the FactorBase design for building CT and CPTs. For this we refer to the unielwin dataset (O. Schulte 2015). In this paragraph we first picture the unielwin entity relationship diagram, then we explain the concepts or parameterized factors and show how the CTs and CPTs are being generated. We will not elaborate any further on additional FactorBase logic. Our main contribution as described in the next chapter is for this part of the FactorBase logic, hence we do not explain FactorBase beyond the CT and CPT generation logic.

4.1 Unielwin entity relationship diagram

The unielwin database model is pictured in an Entity Relationship Diagram (ERD) as:

There are three so called entity tables: student, prof and course. An entity table is defined as a table that is used to store the data of a certain entity. The connecting tables that connect the entity tables are defined here as a ‘bridge’ table. In the above picture these are the tables: RA and registration.

4.2 Parameterized factors

A parameterized factor represents an interaction among parameterized random variables, or par-RVs for short. A factor is the basic unit of representing uncertainty and correlations in most of the work on probabilistic graphical models (Pearl 1998). Factors are
functions over small sets of RVs that map each joint instantiation to real numbers between 0 and 1. In FactorBase the ERD is translated to par-RVs (Olivier Schulte 2015, page 4). Two types of par-RVs are defined: attribute par-RVs that correspond to columns in a single table and relationship par-RVs that correspond to relationships between various tables. The advantage of including a relationship par-RV in the model allows the model to represent uncertainty about whether a relation between two entities exists or not.

An atom is an expression of the form \( r(T_1, \ldots, T_n) \) where each \( T_i \) is either a constant or a first-order variable. If all of the \( T_2, \ldots, T_n \) are constants it is a ground-atom or RV, otherwise it is a first-order atom or par-RV. A par-RV is instantiated to an RV by grounding a constant of the appropriate domain for each first-order variable. A par-factor is a pair \( \Phi = (A, \Phi) \), whereby \( A \) is a set of par-RVs, and \( \Phi \) is a function from the values of the par-RVs to the non-negative real numbers. A grounding of a par-RV represents a set of RVs that interact with each other. A set of par-factors defines a joint probability distribution over the ground par-RVs. (Olivier Schulte 2015, page 2).

4.3 CT generation

For SRL there are a number of formalisms defined for describing par-factors (Kimmig 2015). First-order probabilistic graphical models are popular in both SRL and the database community (Kimmig 2015). The model structure is defined by edges in the graph connecting par-RVs, for instance using a parameterized Bayesian Network. A par-factor in a Bayesian Network is associated with a family of nodes. With a family of nodes, we refer to a child node and all of its parents. In a Bayesian Network the factor value represents the conditional probability of the child value node, given its parents node values.

FactorBase is using a count database to store sufficient statistics for the data. For the CTs the following applies: Consider a list of fixed par-RVs. A query is of the form \( \text{(variable = value)} \) pairs. When executing the query, the result set of a query is the set of instantiations of the logical variables such that the query results to true. Every set of par-RVs has an associated CT. This table contains a row for each of the possible assignments of values for the set of par-RVs and a column count to record the count of the query.

For the unielwin ERD diagram, FactorBase will generate the CT tables using the following generic SQL query for the entity tables:

```
CREATE TABLE <table> (column list) AS
SELECT DISTINCT COUNT(*) AS MULT, (column list)
FROM <table>
GROUP BY (column list)
```

*Figure 4.2 The SQL for the creation of a CT for a list of variables*
For example, the CT table for the table `prof` shows the following:

<table>
<thead>
<tr>
<th>MULT</th>
<th>popularity</th>
<th>teachingability</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

CT table output for the table `prof` in the uniwelwin dataset

The result is based on the following generated SQL-query in FactorBase:

```sql
SELECT DISTINCT COUNT(*) AS MULT, 
popularity, teachingability 
FROM prof 
GROUP BY popularity, teachingability
```

*Figure 4.3 SQL query to generate the CT for the relation prof in the uniwelwin dataset*

The primary- and foreign key fields of the table are omitted. Using the GROUP BY in SQL together with the aggregate function COUNT(*) we are able to generate statistics on the frequencies of combined values available in the table `prof`. Each entity table will have a CT table similar to the one shown above for the table `prof`. In our uniwelwin example we end up with three CT tables for the relations `prof`, `student`, and `course`. These newly created tables are called `prof_counts`, `student_counts`, and `course_counts`.

The next step is generating the CTs for the relationship par-RVs. Instead of looking at a single table we now have to generate statistics for multiple tables at the same time. In this example we take the relation `prof / student`. The generic template for the SQL query is:

```sql
SELECT DISTINCT COUNT(*) AS MULT, (column list) 
FROM <table list> 
WHERE (join-clause) = (join-clause) 
GROUP BY (column list)
```

*Figure 4.4 Generic SQL query to generate a CT for a relationship par-RV*

When we apply the generic template SQL query for the `prof / student` relation we get SQL such as:

```sql
SELECT DISTINCT COUNT(*) AS MULT, 
p.popularity, p.teachingability, s.intelligence, 
s.ranking, r.capability, r.salary 
FROM student s, prof p, RA r 
WHERE p.prof_id = r.prof_id and s.student_id = r.student_id 
GROUP BY p.popularity, p.teachingability, s.intelligence, 
s.ranking, r.capability, r.salary;
```

*Figure 4.5 SQL query to generate a CT for relation prof-student in the uniwelwin dataset*
Exploiting relational database technology for statistical machine learning in FactorBase

The result of the above query is:

```
+--------+---------+-------------+---------+--------+-----------+------+
| MULT   | popularity | teachingability | intelligence | ranking | capability | salary |
+--------+---------+-------------+---------+--------+-----------+------+
|    1   |    1    |    2        |    2    |    1   |    med    |
|    1   |    1    |    2        |    2    |    2   |    med    |
|    1   |    1    |    2        |    2    |    3   |    low    |
|    1   |    1    |    2        |    3    |    1   |    high   |
|    1   |    2    |    2        |    2    |    1   |    low    |
|    1   |    2    |    2        |    3    |    5   |    high   |
|    1   |    2    |    2        |    4    |    3   |    med    |
|    2   |    2    |    2        |    1    |    4   |    high   |
|    1   |    2    |    3        |    1    |    2   |    low    |
|    1   |    2    |    3        |    2    |    5   |    high   |
|    1   |    2    |    3        |    2    |    4   |    high   |
|    1   |    2    |    3        |    3    |    1   |    low    |
|    1   |    2    |    3        |    3    |    1   |    low    |
|    1   |    2    |    3        |    4    |    3   |    high   |
|    1   |    2    |    3        |    5    |    1   |    med    |
+--------+---------+-------------+---------+--------+-----------+------+
```

Output of the SQL query that generates count values for the prof/student relation in the unielwin dataset

This is however the first step in generating a CT for the prof / student relation data in the unielwin dataset. FactorBase executes a sequence of steps to generate the final CT table for the prof / student relation. We omit the SQL and table output for all steps executed by FactorBase but instead, we supply an overview of how the relationship par-RVs are represented using CTs. The steps are described using the following pictures:

```
select distinct count(*) as MULT, p.popularity, p.teachingability, s.intelligence, s.ranking, r.capability, r.salary
from student s, prof p, RA r
where p.prof_id = r.prof_id and s.student_id = r.student_id
GROUP BY p.popularity, p.teachingability, s.intelligence, s.ranking, r.capability, r.salary;
```

```
SELECT DISTINCT MULT as MULT, fields from prof and student
FROM a_counts
GROUP BY fields from prof and student
```

```
SELECT prof.MULT ' student.MULT
AS MULT, fields from prof and student
FROM prof_counts, student_counts
```

Figure 4.6 First step for generating the CT for the relation prof-student in the unielwin dataset
In the above figure the relation prof / student is replaced by calling this relationship ‘a’. The SQL generates an a_counts table that holds the counts for the join over the tables prof, student and RA (RA is the bridge table that connects the tables prof and student in the database). Using the SQL aggregate function SUM() and grouping on the prof and student table columns we get the totals for the frequencies in the table ‘a_counts’, which are then stored in the intermediate table ‘a_flat’. A join on the tables prof_counts and student_counts results in a new intermediate table ‘a_star’ representing the Cartesian product of the relation between the tables prof and student. The next step is pictured as:

Figure 4.7 Second step for generating the CT for the relation prof-student in the unielwin dataset

In the second step FactorBase will execute a sort-merge on the data of the tables ‘a_flat’ and ‘a_star’. The data is sorted and then merged. Because the input is sorted the merge join algorithm gets a rows from each input stream (one for each input table) and compares them based on the primary keys. The rows are returned if they are equal. If they are not equal, the lower-value is discarded and another row of that input stream is obtained from that input. This process repeats until all rows have been processed (Knuth 1998). The result of the sort-merge is a table called ‘a_false’ that holds the students that do not have a relation with a professor and vice versa. Important to note here is that the sort-merge is executed in the Java VM and not in the database. The new framework will replace this sort-merge step with a SQL outer join executed on the database engine (see 5.5.2).

An example result of a SELECT * on a table ‘a_false’ for the unielwin dataset can be pictured as:

<table>
<thead>
<tr>
<th>MULT</th>
<th>popularity(prof0)</th>
<th>teachingability(prof0)</th>
<th>intelligence(student0)</th>
<th>ranking(student0)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>[...]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>16</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>16</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>21</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>24</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>5</td>
</tr>
</tbody>
</table>

21 rows in set (0.00 sec)

Example output of the table ‘a_false’ after the sort-merge of ‘a_star’ and ‘a_flat’
Exploiting relational database technology for statistical machine learning in FactorBase

The final step is to create the CT table for the prof / student relation using the table ‘a_counts’ which represents the statistics for the existing prof / student relations and the table ‘a_false’, which represents the missing relations between prof and student. The two tables are merged using a SQL UNION statement. This step can be pictured as:

```
SELECT a_counts.MULT, fields FROM a_counts
UNION
SELECT a_false.MULT, fields FROM a_false
```

![CT Table](image)

Figure 4.8 Final step for generating the CT for the relation prof-student in the unielwin dataset

The above three steps are executed for each relation in the database that is being examined and processed using FactorBase.

An example result of a SELECT * on a table ‘a_CT’ for the unielwin dataset can be pictured as:

```
<table>
<thead>
<tr>
<th>MULT</th>
<th>popularity</th>
<th>teachingability</th>
<th>intelligence</th>
<th>ranking</th>
<th>capability</th>
<th>salary</th>
<th>a</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2</td>
<td>2</td>
<td>4</td>
<td>N/A</td>
<td>N/A</td>
<td>F</td>
<td>-</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>med</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>med</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>T</td>
</tr>
<tr>
<td>16</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>N/A</td>
<td>N/A</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>21</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>N/A</td>
<td>N/A</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>24</td>
<td>2</td>
<td>3</td>
<td>5</td>
<td>N/A</td>
<td>N/A</td>
<td>F</td>
<td>T</td>
</tr>
</tbody>
</table>
```

44 rows in set (0.01 sec)

Example data of the CT table called ‘a_CT’ for the relation prof / student

### 4.4 CPT generation

When FactorBase is deriving predictions from the model it requires that it can estimate values for its parameters. The data likelihood is maximized, which is one of the basic parameter estimation methods for Bayesian Networks (see 2.4.7). The maximum likelihood estimates are equivalent to the observed frequencies of a child value given its parent values. For this FactorBase will use CTs. The CPTs are generated using SQL. Given the statistics in a CT, a CPT containing the maximum likelihood can be generated using aggregation in SQL as in the example below:
The above type of SQL query (Olivier Schulte 2015, page 6) is generated for each and every relationship between two tables. For example, for the prof / student relation via the table RA, FactorBase will generate a CPT for the fields capability and salary (stored in RA). For the course student / course, FactorBase will generate CPTs for the fields grade and sat (stored in registration). Finally, FactorBase will generate CPTs for all fields in the entity tables too. For example, from the table student there are CPTs generated for the columns intelligence and ranking.

An example output of the CPT for the relation prof/student can be pictured as:

```
mysql> select * from a_CP
-> ;
+---+-----------------+---------+-----------+----------+
| a | CP         | MULT    | local_mult| likelihood|
+---+-----------------+---------+-----------+----------+
| T | 0.109649 | 250     | 25        | -55.26   |
| F | 0.890351 | 2030    | 203       | -23.58   |
+---+-----------------+---------+-----------+----------+
2 rows in set (0.01 sec)
```

Example output for the CPT of the relation prof/student.

For each column a CP table is generated and stored in the database. For example, the column ‘rating’ from the table course we can picture the output as:

```
mysql> select * from `rating(course)_CP`
-> ;
+-----------------+---------+-----------+----------+
| rating(course) | CP      | MULT      | local_mult| likelihood|
+-----------------+---------+-----------+----------+
| 1               | 0.300000| 684       | 3         | -3.61     |
| 2               | 0.700000| 1596      | 7         | -2.50     |
+-----------------+---------+-----------+----------+
2 rows in set (0.00 sec)
```

Example output of the CPT for the column rating in the table ‘course’.

The column ‘rating’ shows the different values stored in the table ‘course’. The column ‘CP’ shows the frequency of the values stored. In the output we see 0.30 with a value of ‘1’ and 0.70 with a value of ‘2’. The column ‘MULT’ shows the summation of the parent values from the CT table that contains the overall values for all columns as generated.
The column ‘local_mult’ is the number of occurrences for that value. The column likelihood is calculated by using the SQL log() function on the result of the multiplication (CP * local_mult). The SQL log() function returns the natural logarithm of the passed in number. The likelihood data is added during CPT generation and used for structure learning later on.

Both the CTs and CPTs generated by FactorBase are input for the model score computation and final model selection. We will not elaborate any further on the FactorBase logic because our contributions described in the next chapter will focus on the creation of CTs and CPTs only.
5. Dynamic Views for learning from multi-relational data

In this chapter we describe the major contribution of our work with respect to enhancing multi-relational learning using FactorBase. As described in the high-level overview of FactorBase the execution of the program on a dataset consists of multiple steps. The majority of the time is spent on analyzing the data and constructing the CPT and CT tables (see 7.1 for the details). By using dynamic views, we show that the time spent on creating CPT and CT tables is reduced and at the same time that the various steps that copy data are no longer necessary. This allows multi-relational learning on data that is changed while being learned at the same time in less time.

5.1 Dynamic views

A view (see 2.7.1) can be constructed dynamically using the meta-data stored in so called database system tables. These system tables are database engine specific and contain information about objects stored in the database, data-types used and all kinds of meta-data related to available indexes, relational constraints etc. As an example we refer to the system tables of the database-engine MySQL (MySQL, MySQL 5.7 Reference Manual / Information_schema tables 2016). Constructing a view dynamically involves querying the system tables, generating the SQL-strings that finally result in a CREATE VIEW statement using the tools available in the database. This CREATE VIEW statement is then executed using a prepared statement (see 2.7.2).

5.2 What to create dynamic views from

When learning data using FactorBase various queries are used (see Figure 2.22 and Figure 2.23) to create CT and CPTs. These queries are generated dynamically and used to create several intermediate tables in various physical database schemas and finally result in the CPTs and CTs needed. These tables are then exported to so called csv files that can be used to pass into a Bayesian Network learner library. Picture-wise we can represent this as:

![Figure 5.1 FactorBase and the concept of .csv files](image)
The csv files are generated by exporting the data for the CPT and CT tables for a certain dataset. Our contribution involves the dynamic creation of views based on the (meta) data stored in the database and the relations defined between that data. Depending on the data and data-model multiple views will be created resulting in the necessary CPT and CT virtual tables specifically for the input dataset. No data is copied and the original data set stays intact. The csv files are then generated from the available views directly.

5.3 Advantages of views over base tables in FactorBase
The queries used in FactorBase that result in CTs and CPTs generate various intermediate tables storing intermediate data. Data is then copied from one database schema to the next one and processed further. Finally, the CTs and CPTs have their final layout and can be used to export the data for the csv files. The main limitation from this approach is that data is copied and stored multiple times. For small datasets this is not an issue but with large datasets (gigabytes) this no longer a sustainable option. Another limitation is that because of the copying the original data cannot be changed until the execution of learning the network is completed. Using views instead of copying data around in various schemas we can overcome the described limitations. Views do not store any actual data and when the underlying tables are changed, the changes are directly visible through the view(s).

5.4 Creating dynamic views using prepared statements
A dynamic view is created by accessing the system tables of the database used and generating the SQL statement for the ‘CREATE VIEW’ statement dynamically. The generated SQL statement is then executed causing the query to be stored in the system tables and making the view available for other SQL statements. To illustrate this, we will use an example that shows how a view for a count table is created for a base table stored in the database. We assume for this example that the base table name (called ‘acc’ in this example) is previously retrieved from the system tables. The SQL code to generate the view statement could look:
The sequence of statements generates a SQL-string in the variable @SQLQUERY by adding the actual create view syntax, generating a column-list by selecting the data from the MySQL system tables for the passed in table name and adding the necessary FROM, GROUP BY and ORDER BY clauses. The actual query being executed is:

```
CREATE VIEW acc_counts AS
  SELECT DISTINCT COUNT(*) AS MULT, frequency
  FROM acc
  GROUP BY frequency
  ORDER BY MULT
```

After executing ‘EXECUTE stmt1’ the view is available in the database and can be queried. Below we refer to an example from the data set Financial_std and a table in that data set called ‘acc’. This table has two columns ‘account_id’ and ‘frequency’ and has 409 rows stored.
When we execute ‘SELECT * from acc’ all 409 rows are presented in the output:

```
mysql> select * from acc
    -> ;
+------------+--------+
| account_id | frequency|
+------------+--------+
|      19    | Monthly |
|      37    | Monthly |
|      38    | Weekly  |
| ...        |
|     11328  | Monthly |
|     11349  | Weekly  |
+------------+--------+
409 rows in set (0,00 sec)
```

Result set of a select of the base table acc

After creation of the view for the new count table we can query the view:

```
mysql> select * from acc_counts
    -> ;
+--------+--------+
| MULT   | frequency|
+--------+--------+
|      19 | afterTrans |
|      63 | Weekly    |
|    327 | Monthly   |
+--------+--------+
3 rows in set (0,00 sec)
```

Result set of a select on the view that represents a CPT for the base table acc

We can see that there are 3 different values for the column ‘frequency’, these are counted and grouped resulting in a count table (CPT) for the table ‘acc’ in the dataset Financial_std.
5.5 Implemented framework
In this paragraph we describe the implemented framework for the dynamic views. We first describe the design of the implemented framework, followed by the algorithms used to generate CTs and CPTs based on the data-model of the target database we want to examine. Finally, we explain how the framework was validated followed by a summary of the benchmark results.

5.5.1 Framework design
In this paragraph we describe the design of the framework / architecture of the dynamic view implementation. The framework / architecture can be pictured as:

![Framework diagram]

The data and data-model in the database are examined and learned using a sequence of stored procedures when the client program executes the SQL statement: ‘EXEC LearnDataModel_CT “<database name>” ‘ (1), we assume here that ‘EXEC’ is the syntax to execute a stored procedure. The stored procedure will start processing the meta-data stored in the database-engine to learn the data-model. Various intermediate steps will result in a set of views generated as explained in the algorithm for CT creation (paragraph 5.5.2). These views representing the CTs are then stored in the database (2). The next sequence of stored procedures start with the command ‘EXEC LearnDataModel_CPT “<database name>” ‘, will use the CT-views to generate the next set of views representing the CPTs. These views are also stored in the database (3). The views for the CTs and CPTs are now available for the client program (4) to extract data from. This data is then used in the next step, scoring and selecting a model (5).
5.5.2 Dynamic view algorithm for CTs

The algorithm below describes the pseudo code for building dynamically views that represent the CT tables for the various tables and relations between tables in the database.

**Input:** Database $D$ with set of entity tables, $E_1,...,E_e$ and $R_1,...,R_r$ relationship tables.

**Output:** A set of views $V_1,...,V_v$ that represent the CTs for the entity tables and the relation(s) between entity tables and a set of views $Z_1,...,Z_z$ that represent the CTs over the set of views $V_1,...,V_v$.

1: for $i=1$ to $e$ do
2:   Generate SQL for each entity table $E_i$ and create a view representing the CT for $E_i$
3: end-for
4: for $i=1$ to $r$ do
5:   Generate intermediate views $R_i_{flat}$ and $R_i_{star}$ for each $R_i$ based on $R_i$ and its relation(s) with the entity tables from the set $E_1,...,E_e$
6:   Generate an intermediate view $R_i_{false0}$ on the intermediate views: $R_i_{flat}$ and $R_i_{star}$.
7:   Generate view $R_i_{false}$ by selecting the data from $R_i_{false0}$ and exclude the rows with a count of 0
8:   Generate view $V_i$ for the CT of the relationship table $R_i$
9: count the number of $V_i$ views generated in v
10: end-for
11: for $i=1$ to $v$ do
12:   Generate intermediate views $V_i_{flat}$ and $V_i_{star}$ for each $V_i$ created at line 8 and entity tables $E_1,...,E_e$ and relationship tables $R_1,...,R_r$ in $D$
13:   Generate an intermediate view $V_i_{false0}$ on the intermediate views $V_i_{flat}$ and $V_i_{star}$
14:   Generate view $V_i_{false}$ by selecting the data from $V_i_{false0}$ and exclude the rows with a count of 0
15:   Generate view $Z_i$ for the relationship CT $V_i$
16: end-for
17: return \(\{V_1,...,V_v, Z_1,...,Z_z\}\)

*Figure 5.5 Dynamic view algorithm for CT creation in a database*

Using the above algorithm, the CT tables are generated and stored in the database. The generation of the list of entity tables at is done by querying the system tables in the database engine. All tables with primary keys not having a foreign key relationship are considered to be entity tables. At line 2 the SQL aggregate function COUNT() is used for each entity table using SQL code generated that creates a view for each entity table, similar to query in Figure 2.22. At line 4 the list of relationship tables is created. The system tables of the databases engine are queried and all tables having a foreign key relation (or multiple foreign key relations) are added to this list. At line 5 the intermediate views $R_i_{flat}$ and $R_i_{star}$, are generated using the SQL as pictured in Figure 4.7. At Line 6 the intermediate
Exploiting relational database technology for statistical machine learning in FactorBase

view $R_{i\cdot false0}$ uses a LEFT OUTER JOIN that replaces a sort-merge on the intermediate views $R_{i\cdot flat}$ and $R_{i\cdot star}$. This LEFT OUTER JOIN is necessary to replace the sort-merge as described in Figure 4.7. Line 7 is used to filter out the rows with a count of 0. The final CT table for the relationship table $R_i$ is created using an SQL query similar to what is pictured in Figure 4.8. At line 12 we repeat the process we did earlier at line 5 but now for all the entity tables and relationship tables together. The algorithm assumes that the database allows nested views (i.e. views generated on top of already existing views).

5.5.3 Dynamic view algorithm for CPTs

The next algorithm shows the pseudo code for building dynamically views that represent the CP tables for the various tables and relations between tables in the database.

Input: Database $D$ with set of views $V_1...V_j$ and $Z_1...Z_z$ representing the CT tables for the data and data-model stored in $D$. A list of all non-primary key/foreign key columns $C_1...C_c$ in $D$

Output: A set of views $CP_1...CP_v$ that represent the CPTs for the entity tables, relation(s) between entity tables and all non-primary key columns.

1: for $i=1$ to $c$ do
2: select the CT $Z_j$ table from $Z_1...Z_z$ that holds the overall CT data for column $C_i$
3: calculate the summation $S_c$ of the parent values for $C_i$ from $Z_j$
4: calculate frequency $CP_c$ for $C_i$ using $S_c$ and the value of MULT in $Z_j$ for $C_i$
5: calculate the likelihood $L_c$ by executing the SQL log() function on
   ( $CP_c$ * value frequency )
6: Generate SQL for each $C_i$ and create a view $CP_i$ representing the vector values for $C_i$ and the values for the calculated $S_c$, $CP_c$ and $L_c$
7: end-for
8: return $\{CP_1...CP_v\}$

Using the algorithm for CPT creation (see paragraph 5.5.3) the CPTs for all columns and their values are calculated. Previously we described what a CP table looks like and how the various columns are being calculated. With the selected CT table from line 2 all information on value-vectors, value-frequencies are available for the processed column $C_i$. In line 4 $CP_c$ is calculated dividing MULT from $Z_j$ by the calculated $S_c$, in other words: the frequency-count for a column value divided by the total frequency count of the parent values. In line 6 the SQL syntax is generated based on the calculations and the available meta-data in the database for the column $C_i$ being processed. With the SQL generated a CPT for column $C_i$ is generated and stored in the database.
To illustrate what type of SQL the new algorithm generates we show the output of the queries generated for the CPT view for an example column called ‘teachingability’ from the unielwin data set:

```
CREATE VIEW teachingability_CP0 AS
SELECT teachingability, COUNT(*) AS local_mult,
( SELECT COUNT(*) FROM prof ) as total,
( SELECT SUM(MULT) FROM ra_prof_student_CT
WHERE ra_prof_student_CT.teachingability=prof.teachingability ) AS parent_value
FROM prof GROUP BY teachingability

CREATE VIEW teachingability_CP1 AS
SELECT teachingability, 0 AS local_mult,
0 as total,
SUM(MULT) AS parent_value
FROM ra_prof_student_CT
WHERE ra_prof_student_CT.teachingability = 'N/A'

CREATE VIEW teachingability_CP2 AS
SELECT teachingability, local_mult/total AS CP,
parent_value, local_mult,
log(local_mult/total * local_mult) as likelihood
FROM teachingability_CP0
UNION
SELECT teachingability, 1.0/1.0 AS CP,
parent_value,
local_mult, 0.0 as likelihood
FROM teachingability_CP1

CREATE VIEW teachingability_CP AS
SELECT * FROM teachingability_CP2
WHERE teachingability is not NULL
```

```
mysql> select * from teachingability_CP
+-----------------+--------+--------------+------------+-------------+
| teachingability | CP     | parent_value | local_mult | likelihood  |
|-----------------+--------+--------------+------------+-------------+
| 2               | 0.50000| 114          | 3          | 0.4054651081081644 |
| 3               | 0.50000| 114          | 3          | 0.4054651081081644 |
+-----------------+--------+--------------+------------+-------------+
2 rows in set (0.00 sec)
```

Example output for a CPT generation run using the new algorithm for CPT creation

5.5.4 Framework validation

The result of running the stored procedures is a set of views in the current database. The database is previously created using the dataset as downloaded (O. Schulte 2015). After running the stored procedures, the views for the CTs and CPTs are available. The validation was done using two mechanisms: (1) by comparing the results of FactorBase stored in csv files in a dataset specific directory with the contents of the views created by the framework implemented. (2) exporting the data towards files in a target directory and comparing the generated files using the Unix diff command. The comparison was executed to make sure the new and enhanced logic did generate exactly the same results FactorBase was generating.
Exploiting relational database technology for statistical machine learning in FactorBase

For example, when we run FactorBase for the unielwin dataset, a directory gets populated on the system running FactorBase with the csv files specifically for the unielwin dataset.

This could directory could look as:

```
Invictus:csv reneq6 ls
a,b.csv    a.csv    a_false.csv bTrue.csv prof0.csv
a,bTrue.csv aTrue.csv b.csv    course0.csv student0.csv
```

Directory output listing for the unielwin dataset

The csv files represent the CTs for the tables course, prof and student. The file a.csv represents the relation between prof and student and the file b.csv represents the relation between course and student. The file a_false.csv and b_false.csv represent the non-existent relations between the prof / student tables and the course / student tables. The a,b csv files represent the data about the relation prof /student and course/student together.

Using the csv files and the views created in the database we can compare the data. As an example we take course0.csv. The file course0.csv has the following contents:

```
MULT    diff    rating
1       1       1
2       2       1
3       2       2
4       1       2
```

Content of the course0.csv file of the unielwin dataset

Also the database scheme is extended by FactorBase adding the table course0_counts for the same data. The content of this table is:

```
mysql> select * from course0_counts
    -> ;
+-------------+-------------+-------------------+
| MULT | diff(course0) | rating(course0) |
+-------------+-------------+-------------------+
|    1 | 1           | 1              |
|    2 | 2           | 1              |
|    3 | 2           | 2              |
|    4 | 1           | 2              |
+-------------+-------------+-------------------+
4 rows in set (0,00 sec)
```

SQL result for a SELECT statement on the course0_counts table for the unielwin dataset
Exploiting relational database technology for statistical machine learning in FactorBase

When we compare this with the view generated by the new framework that represents the same type of information on the course table we get:

```
mysql> select * from course_counts
-> ;
```

```
+----------+---------------+-----+
| MULT | rating | diff |
|----------+---------------+-----|
| 1 | 1 | 1 |
| 2 | 1 | 2 |
| 3 | 2 | 2 |
| 4 | 2 | 1 |
+----------+---------------+-----+
```

4 rows in set (0.01 sec)

SQL result for a SELECT statement on the generated view course_counts

The CPTs created by FactorBase are not stored in .csv files but are only stored in the database. For example, a CPT created by FactorBase is:

```
mysql> select * from `ranking(student0)_CP`
```

```
+-----------+-----------------+-------------+-------------------+----------+-----+
| MULT | ranking(student0) | a | intelligence(student0) | ParentSum | local_mult |
|-----------+-----------------+-------------+-------------------+----------+-----|
| 420 | 1 | F | 3 | 580 | 42 | 0.724138 | -13.56 |
| 60 | 1 | T | 3 | 80 | 6 | 0.750000 | -1.73 |
| 250 | 2 | F | 2 | 650 | 25 | 0.384615 | -23.89 |
| 160 | 2 | F | 3 | 580 | 16 | 0.275862 | -20.61 |
| 50 | 2 | T | 2 | 130 | 5 | 0.384615 | -4.78 |
| 20 | 2 | T | 3 | 80 | 2 | 0.250000 | -2.77 |
| 350 | 3 | F | 2 | 650 | 35 | 0.538462 | -21.67 |
| 70 | 3 | T | 2 | 130 | 7 | 0.538462 | -4.33 |
| 320 | 4 | F | 1 | 800 | 32 | 0.400000 | -29.32 |
| 50 | 4 | F | 2 | 650 | 5 | 0.076923 | -12.82 |
| 40 | 4 | T | 1 | 480 | 4 | 0.000000 | 0.00 |
| 10 | 4 | T | 2 | 130 | 1 | 0.076923 | -2.56 |
| 480 | 5 | F | 1 | 800 | 48 | 0.600000 | -24.52 |
|-----------+-----------------+-------------+-------------------+----------+-----|
```

13 rows in set (0,00 sec)

SQL result for a SELECT statement for the CPT for the column ranking in the table student

For all datasets the tables created by FactorBase were compared with the contents of the views created by the implemented framework. The content of the tables was exported to a file using a SELECT query with the option OUTFILE. For example:

```
mysql> select * from course0_counts into outfile '/tmp/course0_counts'
```

```
Results in:
Invictus:csv reneq$ cat /tmp/course0_counts
1 1 1
2 2 1
3 2 2
4 1 2
```

Unix cat output showing content for course0_counts

This mechanism is then used for all tables involved and the unix diff command compares the created output files.
6. Reducing data movement and query complexity when learning from multi-relational data

In the past the primary method for learning from data was to construct hypotheses and test the hypotheses by analyzing data. Today, learning from data using automated tools is one of the new ways of learning from data. The two main steps for learning from data are: modelling and scoring as explained earlier. In general modelling develops rules from analyzing the data. The rules are later used to examine new cases. The model that is generated from the rules is then used to make predictions via scoring. The numerical values of the scores represent the certainty of each prediction. There are basically two main methods to implement model scoring in an application. One method involves loading the data from a database into a model using SQL and for example Open Database Connectivity (ODBC). The scoring is executed where the model is stored and the results are stored in the database. Another method is to use a client application (like FactorBase) written in Java for example, to represent the model. The application runs against the database that stores the data. Data is moved from the database to the application, processed and maybe changed and then moved back into the database.

The most expensive operations involving storing and accessing data are seeks on the hard disk the data is stored on. In order to improve performance, related data should be stored such that the number of seeks is minimized. This is also known as locality of reference. Hard disks are organized into series of blocks of a certain size. By organizing the table data such that rows fit within these blocks, and grouping related rows onto sequential blocks, the number of blocks that need to be accessed is minimized, along with the number of seeks. Database engines are designed to efficiently return data for an entire row in as few operations as possible. By storing the row’s data in a single block on the disk, along with related rows, the database engine can quickly retrieve the rows with a minimum of disk operations.

6.1 Building models inside the database

When running a test for a dataset using FactorBase we notice that in total there are 5 different database schemas being used to run one single test. Tables are populated using the initial scripts (O. Schulte 2015), from there data is analyzed, changed and copied into various other database schemas. For small datasets this is not a problem, but with large datasets this approach is not longer feasible because the copying takes time but also requires additional disk space. The basic idea of building models inside the database is the most efficient because model(s) and data are kept together. To build models the data in tables is analyzed and statistics are gathered and stored that show information on the data values available. In our example we will focus on a scenario whereby the data value counts (number of times a certain value is present in a column) and the data frequencies are calculated and stored. These are typical data properties used in a multi-relational learning application like FactorBase. The approach is as follows: (1) determine the total number of rows in the table, (2) For each column in the table execute a SQL SELECT using a DISTINCT to retrieve the number of unique data values, calculate the frequency for the values involved. (3) insert the result of (2) into a table that holds the selected data. This means that:
Exploiting relational database technology for statistical machine learning in FactorBase

- All the data in the table is read n-column times. This is because a relational database is a typical row store based engine. When selecting data from disk (or memory when already seen before) a complete row is processed;
- Data in a database is typically stored on memory/disk pages, based on the size of a data row, a certain number of rows fit on a memory/disk page. Accessing the data results in reading / writing pages;
- A subset of the original table data is created and stored (data is duplicated).

The implications of the above are significant. Assume the following scenario:
- table size is 2G (2000 Megabytes) with;
  - 50 columns;
  - we assume the database is capable of holding the complete table in memory;
  - A disk i/o takes 4ms and a memory i/o takes 2ms\(^6\);
  - An average page-size for a database is 16K (default for MySQL for example);
  - No optimizations are done (like indexes, intermediate table etc.);
  - Data is not present in memory when the querying starts.

The following calculation can be applied on the above scenario:
- Initial disk i/o’s needed to read data from disk (2097152000 bytes / 16384 bytes ) = 128000 i/o’s * 4ms = 512 sec.
- 49 times a scan in memory for all data is 128000 i/o’s * 2ms * 49 = 12544 sec.
- disk i/o’s need to write the query results to disk: We assume one row for each column, which is 50 i/o’s * 4ms = 0.2 second;
- This brings us to a total of 512 + 12544 + 0.2 = 13056.2 seconds = 217 minutes to process a table of 2G with 50 columns. In a real case scenario, the database engine will be able to optimize the reading and writing to disk but still the majority of time is spent on selecting all data 49 times to process the data for the 49 different columns in the table.

The above demonstrates that calculating data value counts and frequencies on large data-sets for a table with a significant number of columns is not efficient and far from optimal. In the next paragraphs we describe an algorithm that reduces the number of times data needs to be processed and the number of queries that is used to select the data. Similar to the algorithms discussed in the previous chapter, this additional algorithm is based on generating views dynamically from the database meta data. Using the views, the data can be transformed such that less I/O and queries are needed to achieve the same result.

---

\(^6\) Both disk and memory i/o speeds are based on the current statistics available on the internet for hard-drives and memory modules. As an example for hard drive numbers we refer to: [https://en.wikipedia.org/wiki/Hard_disk_drive_performance_characteristics#SEEKTIME](https://en.wikipedia.org/wiki/Hard_disk_drive_performance_characteristics#SEEKTIME). For memory access numbers we refer to: [https://en.wikipedia.org/wiki/CAS_latency](https://en.wikipedia.org/wiki/CAS_latency).
6.2 Data movement and query reduction algorithm

Below we describe the algorithm to generate dynamic views for data value counts and associated frequencies for all columns in a target database.

**Input:** Database $D$ with set of $T_2,...,T_t$ tables.

**Output:** A set of views $V_j,...,V_v$ that represent the column frequencies for each table in database $D$.

1: for $i=1$ to $t$ do
2:   Generate a list of columns $C_1,...,C_c$ for $T_i$
3:   for $k=1$ to $c$ do
4:     create a view $VC_k$ with the $C_k$ name as ‘attribute name’
5:     and the $C_k$ data as ‘value’
6:     end-for
7:   Generate view $V_{i+1}$ which is a union all of $VC_1,...,VC_k$
8:   Generate view $V_i$ on top of $V_{i+1}$ that displays the attribute data, value,
9:   count of the value and the frequency of the value
10: use a single SELECT to create the data counts and frequencies for all
11: columns $C_1,...,C_c$
12: end-for
13: return $\{V_j,...,V_v\}$

Figure 6.1 Dynamic view algorithm for reducing data movement and reducing query complexity when calculating column frequencies

The above algorithm will use the database meta-data to dynamically generate the CREATE VIEW syntax. Based on the table names and column names views are dynamically created using string concatenation and prepared statements. In paragraph 6.4 we explain the algorithm using an example.

6.3 Implemented framework

The framework used is identical to what is being implemented for the dynamic view algorithms that generate the CTs and CPTs (see paragraph 5.5.1). Again a stored procedure is created that can be called using the SQL EXEC statement. The stored procedure will query the meta-data for the database and uses prepared statements to generate a view that contains the data count values and associated frequencies for each table in the database.

6.4 Algorithm example

The below example is based on logic that is executed in FactorBase while building the CTs and CPTs. The data values for each column in a table are examined to determine the number of distinct values for each data value of that column.
Exploiting relational database technology for statistical machine learning in FactorBase

We demonstrate the algorithm using a table from the unielwin dataset. We take the table ‘RA’ which is the relationship table between the tables ‘prof’ and ‘student’ (see the ERD for in appendix [ref to be done]). The table ‘RA’ has the following layout:

```
mysql> select * from ra
    -> ;
+-----------+-------+-----------+-------+
<table>
<thead>
<tr>
<th>capability</th>
<th>prof_id</th>
<th>student_id</th>
<th>salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>4</td>
<td>17</td>
<td>med</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>5</td>
<td>low</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>14</td>
<td>low</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>18</td>
<td>high</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>26</td>
<td>high</td>
</tr>
</tbody>
</table>
[...]
| 4         | 7     | 8         | high  |
| 1         | 7     | 11        | med   |
| 5         | 7     | 19        | high  |
| 1         | 7     | 20        | low   |
| 3         | 7     | 22        | med   |
| 1         | 8     | 12        | med   |
| 2         | 8     | 13        | med   |
| 3         | 8     | 21        | low   |
| 3         | 9     | 30        | high  |
+-----------+-------+-----------+-------+
25 rows in set (0.01 sec)
```

The algorithm will generate two views in this case. One for column ‘capability’ and one for the column ‘salary’. The generated SQL syntax for the capability columns is:

`CREATE VIEW ra_capability AS SELECT student_id, prof_id, ‘capability’ as attribute_name, capability as value from RA order by student_id, prof_id`  

For the columns salary the syntax is:

`CREATE VIEW ra_salary AS SELECT student_id, prof_id, ‘salary’ as attribute_name, salary as value from RA order by student_id, prof_id`  

Then both views are combined into a final view that is basically a pivot of the original table ‘RA’. This view is created using the following syntax:

`CREATE VIEW ra_capability_salary AS SELECT * from ra_capability UNION ALL SELECT * from ra_salary order by student_id, prof_id`  

The result of a SQL SELECT on the ra_capability_salary view will look as:

```
mysql> select * from ra_capability_salary
    -> ;
+-----------+-------+----------------+-----+
<table>
<thead>
<tr>
<th>student_id</th>
<th>prof_id</th>
<th>attribute_name</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>5</td>
<td>capability</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>salary</td>
<td>low</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>capability</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>salary</td>
<td>low</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>salary</td>
<td>high</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>capability</td>
<td>4</td>
</tr>
<tr>
<td>8</td>
<td>6</td>
<td>salary</td>
<td>high</td>
</tr>
</tbody>
</table>
```
Exploiting relational database technology for statistical machine learning in FactorBase

<table>
<thead>
<tr>
<th>8</th>
<th>6</th>
<th>capability</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>7</td>
<td>salary</td>
<td>high</td>
</tr>
<tr>
<td>8</td>
<td>7</td>
<td>capability</td>
<td>4</td>
</tr>
<tr>
<td>9</td>
<td>6</td>
<td>capability</td>
<td>4</td>
</tr>
<tr>
<td>9</td>
<td>6</td>
<td>salary</td>
<td>high</td>
</tr>
<tr>
<td>10</td>
<td>9</td>
<td>salary</td>
<td>high</td>
</tr>
<tr>
<td>[...]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>7</td>
<td>capability</td>
<td>1</td>
</tr>
<tr>
<td>20</td>
<td>7</td>
<td>salary</td>
<td>low</td>
</tr>
<tr>
<td>21</td>
<td>8</td>
<td>salary</td>
<td>low</td>
</tr>
<tr>
<td>21</td>
<td>8</td>
<td>capability</td>
<td>3</td>
</tr>
<tr>
<td>22</td>
<td>7</td>
<td>salary</td>
<td>med</td>
</tr>
<tr>
<td>22</td>
<td>7</td>
<td>capability</td>
<td>3</td>
</tr>
<tr>
<td>23</td>
<td>6</td>
<td>capability</td>
<td>5</td>
</tr>
<tr>
<td>23</td>
<td>6</td>
<td>salary</td>
<td>high</td>
</tr>
<tr>
<td>24</td>
<td>6</td>
<td>salary</td>
<td>high</td>
</tr>
<tr>
<td>24</td>
<td>6</td>
<td>capability</td>
<td>5</td>
</tr>
<tr>
<td>25</td>
<td>6</td>
<td>salary</td>
<td>high</td>
</tr>
<tr>
<td>25</td>
<td>6</td>
<td>capability</td>
<td>5</td>
</tr>
<tr>
<td>26</td>
<td>5</td>
<td>capability</td>
<td>4</td>
</tr>
<tr>
<td>26</td>
<td>5</td>
<td>salary</td>
<td>high</td>
</tr>
<tr>
<td>27</td>
<td>5</td>
<td>salary</td>
<td>med</td>
</tr>
<tr>
<td>27</td>
<td>5</td>
<td>capability</td>
<td>3</td>
</tr>
<tr>
<td>28</td>
<td>5</td>
<td>capability</td>
<td>3</td>
</tr>
<tr>
<td>28</td>
<td>5</td>
<td>salary</td>
<td>med</td>
</tr>
</tbody>
</table>

50 rows in set (0,01 sec)

SQL output for pivot table of the columns salary and capability

We ‘transformed’ the original data from columns into rows. Using this transformed data it is very simple to retrieve the frequency information for the data in the columns of the table ‘RA’. For that a final view is created using below syntax:

```
CREATE VIEW ra_frequencies as
SELECT distinct(attribute_name), value, count(*) as count, count(*) / (select count(*) from RA) as frequency
FROM ra_capability_salary
GROUP BY attribute_name, value
```

A SELECT from the above view will result in:

```
mysql> select * from ra_frequencies
-> ;
+-----------------+---------+-------+-----------+
| attribute_name  | value   | count | frequency |
|-----------------+---------+-------+-----------|
| capability      | 1       | 5     | 0.2000    |
| capability      | 2       | 4     | 0.1600    |
| capability      | 3       | 7     | 0.2800    |
| capability      | 4       | 5     | 0.2000    |
| capability      | 5       | 4     | 0.1600    |
| salary          | high    | 11    | 0.4400    |
| salary          | low     | 5     | 0.2000    |
| salary          | med     | 9     | 0.3600    |
+-----------------+---------+-------+-----------+
8 rows in set (0,01 sec)
```

SQL frequency and count output for the columns capability and salary
The count and frequency data can be selected and used using a single SELECT statement on the view created specifically for a table (in this example ‘RA’). It is no longer needed to execute a SQL SELECT statement per column and inserting the result set into a table that contains the count and frequencies for the data.

The algorithm for reducing data movement and query complexity (see paragraph 6.2) is basically a method to transpose an existing table using views. The columns are presented / stored as rows allowing queries that need to access all columns one by one with the same query, to be changed into a single query that accesses all column data and generates the same result. The column data is processed in a row format for which the database engine is optimized and designed.
7. Experiments
In this chapter we describe the results of the various improvements for learning from a multi-relational database model using FactorBase. First we start we the baseline test we conducted. The idea behind the baseline test is to have a starting point which later on can be used to validate if the suggested improvements are indeed making any difference.

7.1 Baseline test FactorBase
In this paragraph we describe the results of the baseline test which is conducted on the selected datasets for FactorBase. We will first describe the datasets used, followed by a description of the test environment. We then describe the test setup and how the test-runs were executed. We will finish with a description of the aggregate results and first conclusions. For the baseline details we refer to appendix (11.3).

7.1.1 Datasets
For the baseline test we did select six datasets. In total there are eight datasets available. The selection was done after a trial run for all available datasets. Some datasets could not be used because FactorBase was not able to complete a run without errors. At the time of setting up the baseline test we did not want to make adjustments to the existing FactorBase program nor did we want to manipulate the contents of a dataset to get around the runtime errors. The selected datasets vary enough in size and time needed to execute, to give us an overall impression on where time and resources are spent. The selected datasets are available at: http://www.cs.sfu.ca/~oschulte/BayesBase/input-output.html.

For the baseline test we used the following datasets:

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Total number of tables / relationship tables</th>
<th>Total number of columns for all tables</th>
<th>Total number of rows for all tables</th>
</tr>
</thead>
<tbody>
<tr>
<td>Financial_std</td>
<td>9/3</td>
<td>22</td>
<td>226,935</td>
</tr>
<tr>
<td>Hepatitis_std</td>
<td>7/3</td>
<td>25</td>
<td>12,927</td>
</tr>
<tr>
<td>MovieLens_std</td>
<td>3/1</td>
<td>20</td>
<td>82,333</td>
</tr>
<tr>
<td>MovieLens_TQ</td>
<td>3/1</td>
<td>10</td>
<td>1,008,926</td>
</tr>
<tr>
<td>Mutagenis_std</td>
<td>3/1</td>
<td>11</td>
<td>9,647</td>
</tr>
<tr>
<td>unielwin</td>
<td>5/2</td>
<td>17</td>
<td>171</td>
</tr>
</tbody>
</table>

*Figure 7.1 Baseline datasets used for testing*

The datasets were loaded into the database server using the SQL format as published.

7.1.2 Test environment
The baseline test is conducted using MySQL and Java 1.8 and executed on an Intel quad core i5 processor with a clock speed of 3.3Ghz The operating system used is MacOs 10.12.1 (Sierra). The MySQL server version is 5.6.29 with 8Gig of RAM assigned. For all details on the baseline test environment we refer to the appendix (11.2).
7.1.3 Test setup
The baseline test on the datasets did run from the OS command-line using a terminal session. A test run consists at a high level of loading the data into the database followed by building the necessary tables needed to learn the data model. This results in a new database schema that contains information on the tables, the various relations between tables and the columns and values in the columns. The next step is to create the CT and CPT tables which are also stored in a new database schema. The CT and CPT tables are then exported to CSV files in the FactorBase home-directory. These files can then be used on a spreadsheet program like Excel or Google spreadsheets. After this stage FactorBase will start learning the parameters and the structure of the network. The output of a test run is a Bayesian network that shows probabilistic dependencies between the relationships and attributes represented in the database. The output was stored in two formats:

1. The Bayesian Network model represented in relational tables stored in a separate database schema. It contains the Bayesian Network graph, the conditional probability tables and statistical scores;
2. A Bayesian Network interchange format file. This can then be used in a Bayesian Network reader program.

7.1.4 Test execution
For all datasets we followed the same sequence of steps to execute the test. The MySQL database schemas of the previous runs were dropped and all database monitor counters were cleared. The database engine was restarted and the downloaded SQL files of the datasets were used to create the tables for the dataset and load the data. Before starting a sequence of 10 runs for the same dataset, a warming up run was executed to allow the database engine to cache the data where possible. The output of each run (see example in appendix 11.1) was saved in a text file and the various execution times were stored in a spreadsheet. We did note the time for the following execution steps: Building time, CT building time, Smoothed CP time, CSV pre-computer, Parameter Learning and Structure Learning. For each dataset we did store the average execution for each step, a circle diagram containing the percentages of execution time spend, a description of the dataset followed by the EER diagram representing the relationships between the various tables.
7.1.5 Conclusions

After running the tests for all datasets the average times (in milliseconds) for each test execution step was aggregated into a single spreadsheet and the overall baseline results can then be pictured as:

![Graph showing baseline results for all datasets measured in milliseconds]

Due to the variation in data size and model complexity the times recorded cannot be presented such that we can draw conclusions based on the graph generated. In the above graph the x-axis property was changed to a log with a base of 10 to get a better overview.

Looking at the overall baseline results we see that for two datasets: Hepatitis_std and Financial_std most of the time is spent in Building Time, CT building Time and Parameter Learning. The other steps: Smoothed CP, Structure Learning and CSV pre-computer do not show these peaks in the times recorded for these two datasets. For the remaining datasets the difference between the times for all steps is much smaller. The initial results are a surprise because we were assuming that typically learning the parameters and the structure of the network would take most of the time for all datasets.
When we check the results per dataset (see appendix 11.3) we can see that indeed CT and CPT building time and parameter learning are the major components where time is spent. There seems to be a relation between the number of relationships in the model (checked via the EER diagram) and the amount of data. This is reflected and summarized in the below table:

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Total size in MB</th>
<th>Number of relationship tables in EER</th>
<th>CT/CPT building time percentage</th>
<th>Parameter Learning percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Financial_std</td>
<td>15</td>
<td>3</td>
<td>66</td>
<td>10</td>
</tr>
<tr>
<td>Hepatitis_std</td>
<td>2</td>
<td>3</td>
<td>62</td>
<td>15</td>
</tr>
<tr>
<td>MovieLens_std</td>
<td>13</td>
<td>1</td>
<td>20</td>
<td>44</td>
</tr>
<tr>
<td>MovieLend_TQ</td>
<td>84</td>
<td>1</td>
<td>47</td>
<td>28</td>
</tr>
<tr>
<td>Mutagenesis_std</td>
<td>1</td>
<td>2</td>
<td>24</td>
<td>29</td>
</tr>
<tr>
<td>unielwin</td>
<td>1</td>
<td>2</td>
<td>24</td>
<td>38</td>
</tr>
</tbody>
</table>

*Figure 7.3 Overview of percentage time used for CT and CPT generation for all datasets used*

7.2 Dynamic views for learning from relational data

In this paragraph we describe the results of the dynamic view algorithm. The two steps executed in the dynamic view algorithm consist of building the CTs and the CPTs (also referred to as parameter learning).

7.2.1 Experiment results

The diagrams in Figure 7.5 and Figure 7.7 are used to compare the time needed for building the CTs and CPTs using FactorBase and using the dynamic views algorithms. The diagrams show the various datasets and the time needed (in milliseconds) to execute the particular step. Both diagrams have a time axis using a log 10 to better display the results.

Times are measured using the MySQL performance report tools. For each dataset we did run the creation of CPTs and CTs ten times and then used the average query execution as presented in the MySQL performance report. An example of such a report is pictured below:
The report will list the queries / stored procedures that are being executed together with all kinds of additional information like: number of times executed, number of rows inserted, locks taken, etc. In the diagrams we present later we used the average execution time from the performance report. The time is calculated in microseconds in MySQL but is converted to milliseconds in the diagrams so it is easier to compare with the times measured in FactorBase, which are also calculated in milliseconds.

We first present the results for the creation of the CT views, followed by the results for creating the views that represent the CPTs.
Exploiting relational database technology for statistical machine learning in FactorBase

7.2.1.1 CT building results

The diagram shows that for two datasets (Hepatitis_std and Financial_std) the differences between FactorBase and dynamic views is a factor of 621 for the Hepatitis_std dataset and a factor of 204 for the Financial_std dataset. For the Mutagenesis_std dataset we see a 45% improvement, for the MovieLens_TQ we see an 80% improvement and finally for the uni엘win dataset we see a 27% improvement. For the MovieLens_std dataset the difference is about 1%.

The difference between hepatitis_std and financial_std versus the other datasets is the number of tables and relations between tables involved in the data model. The size of the data seems to be of less interest. For example, the MovieLens_TQ data has by far the largest table (+/- 83Mb) but has a simple data model (see 11.3.4). If we compare number of tables and relations between datasets as shown in Figure 7.6 we can indeed see that this is the main difference.

<table>
<thead>
<tr>
<th>Datasets</th>
<th>Size (in Mb)</th>
<th>Number of columns</th>
<th>Number of tuples</th>
<th>Number of relations</th>
<th>Number of tables</th>
</tr>
</thead>
<tbody>
<tr>
<td>uni엘win</td>
<td>1</td>
<td>17</td>
<td>171</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>Mutagenesis_std</td>
<td>1</td>
<td>11</td>
<td>9647</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>MovieLens_std</td>
<td>13.5</td>
<td>10</td>
<td>82333</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>MovieLens_TQ</td>
<td>83.5</td>
<td>10</td>
<td>1008926</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>Hepatitis_std</td>
<td>2</td>
<td>26</td>
<td>12927</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>Financial_std</td>
<td>15</td>
<td>22</td>
<td>226935</td>
<td>6</td>
<td>7</td>
</tr>
</tbody>
</table>

Both the dataset Financial_std and Hepatitis_std have the most relations (6) and tables (7) in their data model. The number of relations is the major factor that determines how much time is needed to generate the CT data for the tables and columns involved. There seems to
be a relation between the number of CT tables and the number of relations in the data model. Each relation will result in a CT table and after that the CTs themselves are joined and will result in a new CT table (see algorithm for CT creation in paragraph 5.5.2 for details).

7.2.1.2 Parameter learning results

![Parameter Learning time graph]

Parameter learning is basically generating a CPT for each parameter (column, or combination of columns) and storing the data frequency of the data values for that column. The number of columns and the number of rows are therefore the determining factors when looking at the time needed to generate the CPT views.

Again we see that the data sets hepatitis_std and Fincial_std show the most improvements. For the Hepatitis_std dataset we see an improvement with a factor of 64 and for the Financial_std dataset we see an improvement of a factor 18. For the unielwin dataset we see an improvement of 33%. For the Mutagenesis_std we see that FactorBase is faster than the dynamic view algorithm, about a factor 2.5, this also seen for the datasets MovieLens_TQ (1.25) and MovieLens_std (1.28). With these results and looking at Figure 7.6 we see that the datasets with a higher number of columns (financial_std, hepatitis_std and unielwin) show an improvement in time for the dynamic view framework.

7.2.2 Conclusions

Using dynamic views for multi-relational learning offers various advantages compared to the logic being used in FactorBase for generating CTs and CPTs. Using database views allows the data to be changed on the fly and any data change is directly visible in the associated CPT and CT. Also, the creation of the views is a step that is executed once. FactorBase needs to run every time the complete sequence of steps for creating CTs and CPTs when data is changed. Therefore, dynamic views offer a great advantage in a production environment whereby the data set is changed more frequently.

Building dynamic views for CT and CPTs is fast and outperforms FactorBase when a data-model has more tables and relations between tables. From the tests run we can see that this point is reached with 5-6 tables and 4-5 relations.
A fundamental limitation of the dynamic view algorithm proposed, is that it clearly depends on the ability of the database engine to support prepared statements and views that can be build on top of other views, so called nested views. At the same time FactorBase is using very specific SQL syntax that is currently only supported by MySQL. The dynamic view algorithms (see 5.5.2 and 5.5.3) seem to be a more portable solution because the majority of database engines we are aware of are supporting prepared statements and nested views.

7.3 Reducing data movement and query complexity when learning from relational data

To demonstrate the algorithm as described in 6.2, we execute two experiments. The first experiment is on the largest table in the data sets (O. Schulte 2015). This is the table ‘u2base’ from the data set ‘MovieLens_TQ’. This table has 3 columns and has 1000129 rows with a total size of 83.2 Mb. We will not consider the other five datasets because the size of these sets is relatively small and we do not expect to see any benefit of the algorithm for such small sets. Because we want to show the impact of the algorithm (see 6.2) and confirm that this indeed works for larger datasets we use a table of 11 Gig. in size with five columns.

For both experiments we measure the total number of I/O’s and statement execution time using the MySQL performance report tool. The experiment consists of several steps. First we set a baseline result for:

- Determining the total number of rows of the table;
- Calculate the distinct data value count and frequency for each of the columns in the table by executing a single SELECT for each column and insert the data in a target table.

The baseline is run 10 times to generate an average for the I/O’s needed to calculate the data and get an average for the statement execution time. After running the baseline, the next steps are:

- Create the necessary views as described in the algorithm for reducing data movement and query complexity (see paragraph 6.2);
- Run the single SELECT query to determine the distinct data value count and frequency for each column in the table and insert the data in a target table.

Also this last step is run 10 times to generate an average for the I/O’s needed to calculate the data and statement execution time. For each experiment two diagrams are created, one for the total I/O needed and one for the time needed to execute the queries.
7.3.1 First experiment on a small data set

Using the MySQL Workbench tool, we can picture the properties of the u2base table we use for the first experiment:

![MySQL output showing properties of the table used](image1)

For this experiment two diagrams are generated. We first show the number of I/O’s needed (this includes disk read, writes and memory I/o).

![I/O's needed](image2)

The difference on the total I/O’s needed is about 3-4%. The next step is comparing the time needed to execute the queries.
7.3.2 Second experiment on a large data set
To demonstrate the algorithm on a larger data set, we created a data set of 11 Gig. The data sets for FactorBase could not be used to demonstrate the algorithm on a large data set due to their limited size (O. Schulte 2015). The table for the experiment is a table with 5 columns and 36016842 number of rows. Below we show the MySQL workbench screenshot that display the table details:

The output for an example run for the experiment will look as:
Exploiting relational database technology for statistical machine learning in FactorBase

Total number of rows to examine
36016842

*a_priori_counts_Phase*

Generated a priori counts are:

<table>
<thead>
<tr>
<th>attribute_name</th>
<th>value</th>
<th>prior_count</th>
<th>prior_probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>class</td>
<td>N</td>
<td>12863183</td>
<td>0.357</td>
</tr>
<tr>
<td>class</td>
<td>P</td>
<td>23153659</td>
<td>0.643</td>
</tr>
<tr>
<td>Humidity</td>
<td>high</td>
<td>18008443</td>
<td>0.500</td>
</tr>
<tr>
<td>Humidity</td>
<td>normal</td>
<td>18008399</td>
<td>0.500</td>
</tr>
<tr>
<td>outlook</td>
<td>sunny</td>
<td>12863161</td>
<td>0.357</td>
</tr>
<tr>
<td>outlook</td>
<td>rain</td>
<td>12863161</td>
<td>0.357</td>
</tr>
<tr>
<td>outlook</td>
<td>overcast</td>
<td>10290520</td>
<td>0.286</td>
</tr>
<tr>
<td>Temperature</td>
<td>cool</td>
<td>10290542</td>
<td>0.286</td>
</tr>
<tr>
<td>Temperature</td>
<td>hot</td>
<td>10290542</td>
<td>0.286</td>
</tr>
<tr>
<td>Temperature</td>
<td>mild</td>
<td>15435758</td>
<td>0.429</td>
</tr>
<tr>
<td>Windy</td>
<td>false</td>
<td>20581062</td>
<td>0.571</td>
</tr>
<tr>
<td>Windy</td>
<td>true</td>
<td>15435780</td>
<td>0.429</td>
</tr>
</tbody>
</table>

Output of queries that generate the data count and frequency data for the large test set

To measure the number of I/O’s needed and the time to execute the query we used the performance reporting tool which is integrated in MySQL and can be used via MySQLWorkBench to generate reports from. In the below diagrams we present the various results:

![I/O's needed](image)

*Figure 7.12 Comparison of I/O’s needed for baseline and new algorithm for large dataset*

The difference in I/O’s is about 1% which is in practice not significant. When we look at the effect on the execution time we see a different result.
7.3.3 Conclusions
Looking at query time needed, the algorithm based on views transposing the data from columns to rows shows that the algorithm outperforms the standard mechanism of executing a query per table column. For the small table experiment we see an improvement of a factor 5.8. For the second experiment we see an improvement with a factor of 1.2. The major difference in both approaches is that the algorithm allows the database engine to optimize the final SELECT query. The individual SELECTs for each column are difficult to optimize because in order to determine a count of all distinct data values of a column the complete table must be read (we do not take into account that an index on each column would help here). The various views and the union of them will be analyzed by the database engine optimizer together with the single SELECT that calculates the data counts and associated frequency. The optimizer can determine based on the size of the table, the available memory, disk speed etc. which query plan fits best.

The number of I/O’s for the SQL statement per column (the baseline) is almost as high as the I/O’s for the new algorithm. This is due to the lack of indexes on the columns that allow the new algorithm to use the index data instead of retrieving the data from the data pages in the database. An index page in general only stores the values for that specific column and often compression techniques are used to store more data on a single index page. This results in less I/O’s to read the data. The new algorithm should be extended such that indexes are automatically added on the columns needed for the new algorithm to execute faster using less I/O’s. It was not possible during this research to extend the algorithm and develop the code in the timeframe available.

Typically, a database engine that supports a column based store (like Sybase IQ, MonetDB and VectorWise), is optimized to execute queries that work on a per column basis (Harizopoulos, Abadi en Boncz sd). A column-oriented database engine serializes all of the values of a column together, then the values of the next column, and so on. Because of this different way of storing data, indexes are much smaller and more efficient when used to access data. In a column based store each column is often covered by one or multiple
indexes allowing to store all data (via the indexes) into memory. When reading data only the columns needed are read, compared to a row-based system: for the same query the whole row is read. Imagine a table with 200+ columns and one executes a query to select the distinct data value count for a single column. For a column-store only the single column data is read, in a row-store all columns (200+) are read to just access the data value for one single column. It is beyond the scope of this thesis to elaborate further on row-based versus column-based database engines in the context of multi-relational learning, we think however that this is definitely an area that needs further research in future work.
8. Conclusion
In this chapter we present our conclusions based on the results of the experiments conducted. We start with summarizing the answers to the research questions from paragraph 3.1, followed by a description of the limitations applicable for this research. We list the contributions we have made and finish with what we think can be researched and worked on as future work.

8.1 Answers to research questions

- What improvement can be made in the algorithms used to build the CPTs and CTs in FactorBase that result in a better performance for generating the associated CPTs and CTs for the input datasets?

We have designed various algorithms to dynamically generate views from meta-data stored in the database which describes the data-model of an input data set. Using this dynamic approach and utilizing the power of the database-engine, the CPTs and CTs are made available for an input data set after running the logic once. From there any update on the underlying data is directly reflected in the CPTs and CTs. For the generation of CTs the experiments showed improvements ranging from 0.27 up to 621 times faster (see 7.2.1.1). For the generation of CPTs for small datasets, FactorBase turns out to be faster than the new dynamic view algorithm (ranging from 1.25 – 2.5 times). For datasets with more columns the new dynamic view algorithm shows an improvement ranging from a factor 18 to 64 times (see 7.2.1.2).

- RQ1: What improvements can be made for the algorithm used for determining negative relationships (Zhensong Qian 2014), i.e. the lack of a relationship in the data sets, when creating the CTs for these data sets?

Determining negative relationships and reflecting this in the CTs in FactorBase was done using an expensive algorithm that used a sort-merge on two generated tables that described the existing and non-existing relations between two tables in the database. This sort-merge was replaced building dynamically views that are based on using a SQL LEFT OUTER JOIN as described in the algorithm for CT creation in paragraph 5.5.2. The views combined with the LEFT OUTER JOIN eliminate the need for a sort-merge which contributed to the results as described in 7.2.1.1. During our experiments we did not measure how large this specific contribution is.

- RQ2: What improvements can be made for the SQL algorithms used in FactorBase (Olivier Schulte 2015, Appendix) with respect to the performance and resource usage when processing the data sets?

One of the areas that takes time when running FactorBase on a data set is the data movement using various database schemas. The moving and copying of data is one of the complicating factors when running on large data sets. The current data sets are limited in size and can run in a matter of minutes. We developed an algorithm reducing data movement and query complexity (see paragraph 6.2), that again uses views which are
Exploiting relational database technology for statistical machine learning in FactorBase

dynamically build and represent the data in such fashion that the number of queries needed to determine column based statistics can be gathered using a single SELECT / INSERT statement. This compared to a SELECT / INSERT statement for each column in FactorBase that goes together with moving data around to achieve the same result. The new algorithm showed an improvement with a factor ranging form 1.2 up to 5.8 (depending on the data set size).

- RQ3: What is the effect on the performance and resource usage when using a column-store based database engine for the CT and CPT statistics versus a typical row-store database engine?

It was not possible to conduct an experiment using a column-store database engine in the timeframe we had available for this research. However, based on the experiments conducted in chapter 7 we definitely see that a column-store will outperform a row-store based engine because the queries needed for CTs and CPTs are typically on a per column (or group of columns) basis. Determining the number of distinct data values for a column and with that its frequency is typically one the strengths of column stored based database engines (Harizopoulos, Abadi en Boncz sd).

8.2 Limitations

During our research we encountered various limitations which are important to mention in the overall picture. First we list the limitations related to the research itself, followed by the limitations encountered for the software used.

8.2.1 Research related

With respect to the research itself we encountered the following limitations:

- The available data-sets (O. Schulte 2015) were limited to 6 and relatively small in size. This forced us to create a much larger data set to demonstrate the effectiveness of the algorithm to reduce data movement and query complexity (see paragraph 6.2);

- The effect of replacing the sort-merge logic while determining the negative relationships between data is not separately measured but integrated in the overall performance improvements as seen for the algorithm for CT creation (see paragraph 5.5.2). It is therefore not possible to measure how much this optimization did contribute to the overall performance gains.

8.2.2 Software related

With respect to the software used during our research we did encounter the following limitations:

- Changing the FactorBase code turned out to be tedious exercise taking too much time. The code is not easy to change / to extend. It was decided to implement the new algorithms such that the output of the new algorithms could be compared to the output of FactorBase for the same logic. This made it possible to focus on the new algorithms and not lose too much time on changing existing FactorBase code;

- Currently FactorBase can only be used on the database engine MySQL, it uses very specific SQL syntax only supported in MySQL. The enhancements described in this thesis are also developed using MySQL, however due to the nature of the way it was
implemented (using standard SQL, stored procedures, nested views and prepared statements) we believe that can be ported easily towards other database engines. This is however not researched any further in this thesis.

8.3 Contributions
We have created a mechanism to dynamically create views for CTs and CPTs based on the data set loaded in the database. At the same time the views are constructed such that data can be retrieved much faster. Using stored procedures to build the views we moved all the logic into the database engine centralizing the code and execution of it in a single place. This eliminates the need for moving data between the database and the Java client FactorBase.

The code at the client-side for generating CTs an CPTs is greatly reduced in size (one single call to a stored procedure) and at the same time the power of the database engine is fully utilized. Using views has major advantages over the current implementation in FactorBase:
- The logic is executed once and CPTs and CTs are available for any subsequent run;
- Data in the database can be changed on the fly and changes are directly reflected in the CTs and CPTs. The current FactorBase implementation is not capable of handling updates on the data while processing it at the same time.

Using views, we were able to transform data such, that the number of queries used to determine column based statistics can be reduced to a single SELECT-INSERT statement for each table, instead of having a single SELECT-INSERT statement for each column of a table. Again the code is greatly reduced in size and complexity and mainly executed in the database utilizing the power of the database engine being used.

Combining machine learning with a relational database offers a number of possibilities, several of which have been explored in ongoing and previous research. Qian et al. (Zhensong Qian 2014) discuss work related to the creation of CTs and the algebra needed for this. The paper focuses on a virtual join algorithm for computing sufficient statistics that involve negative relationships. We think that the algorithm for CT creation (see 5.5.2) and specifically the logic avoiding expensive sort-merge operations would be beneficial to explore when creating statistics for large data sets. Sing et al. (S. Singh 2013) describe an algorithm to analyze the database catalogs (or meta data) and generate a set of nodes and Bayesian Network structure based on the database catalogs. SQL is used analyze the catalogs and dynamically interpret the data model. The implemented framework as described in this thesis (see 5.5.1) is something we believe can be used to allow learning a data model much easier and more efficient with less code by utilizing the database engine capabilities as described in the framework.
8.4 Future work

Leveraging the newly developed algorithms inside FactorBase is one of the key items that needs to be done in the future. We demonstrated that using dynamic views, simplifying code and making use of the power of the database engine, greatly reduces the time needed to learn from multi-relational data. Based on the conducted experiments we think it is beneficial to explore if a column-store based database engine offers a better solution with respect to performance compared to row-store based database engines. This can however only be researched if more and larger data sets are available. Using database views for CPTs and CTs allows data to be updated while a model is being examined and scored. It could be interesting to research what the effect is of updating data while models are examined and scored at the same time.
9. Bibliography


De Raedt, Luc. "Logical and Relational Learning, from ILP to MRDM (Cognitive Technologies)." (Springer) 2008.


Exploiting relational database technology for statistical machine learning in FactorBase


10. **SQL results and program output Index**

<table>
<thead>
<tr>
<th>Content of the course0.csv file of the unielwin dataset</th>
<th>.......................... 58</th>
</tr>
</thead>
<tbody>
<tr>
<td>CT table output for the table prof in the unielwin dataset</td>
<td>.......................... 44</td>
</tr>
<tr>
<td>Directory output listing for the unielwin dataset</td>
<td>.......................... 58</td>
</tr>
<tr>
<td>Example data of the CT table called ‘a_CT’ for the relation prof / student</td>
<td>.......................... 47</td>
</tr>
<tr>
<td>Example output for a CPT generation run using the new algorithm for CPT creation</td>
<td>.......................... 57</td>
</tr>
<tr>
<td>Example output for the CPT for the relation prof / student</td>
<td>.......................... 48</td>
</tr>
<tr>
<td>Example output of the CPT for the column rating in the table ‘course’</td>
<td>.......................... 48</td>
</tr>
<tr>
<td>Example output of the table ‘a_false’ after the sort-merge of ‘a_star’ and ‘a_flat’</td>
<td>.......................... 46</td>
</tr>
<tr>
<td>Output of queries that generate the data count and frequency data for the large test set</td>
<td>.......................... 76</td>
</tr>
<tr>
<td>Output the SQL query that generates count values for the prof/student relation in the unielwin dataset</td>
<td>.......................... 45</td>
</tr>
<tr>
<td>Result set of a select of the base table acc</td>
<td>.......................... 53</td>
</tr>
<tr>
<td>Result set of a select on the view that represents a CPT for the base table acc</td>
<td>.......................... 53</td>
</tr>
<tr>
<td>SQL frequency and count output for the columns capability and salary</td>
<td>.......................... 64</td>
</tr>
<tr>
<td>SQL output for pivot table of the columns salary and capability</td>
<td>.......................... 64</td>
</tr>
<tr>
<td>SQL result for a SELECT statement for the CPT for the column ranking in the table student</td>
<td>.......................... 59</td>
</tr>
<tr>
<td>SQL result for a SELECT statement on the course0_counts table for the unielwin dataset</td>
<td>.......................... 58</td>
</tr>
<tr>
<td>SQL result for a SELECT statement on the generated view course_counts</td>
<td>.......................... 59</td>
</tr>
<tr>
<td>SQL SELECT result from table unielwin.RA</td>
<td>.......................... 63</td>
</tr>
<tr>
<td>The result query that contains the CREATE VIEW statement</td>
<td>.......................... 52</td>
</tr>
<tr>
<td>Unix cat output showing content for course0_counts</td>
<td>.......................... 59</td>
</tr>
</tbody>
</table>
11. Appendix

11.1 Running the unielwin dataset from the terminal command line using FactorBase.

Command line used to execute the program:

```
java -jar -Xms2048m -Xmx12G -XX:+UseConcMarkSweepGC -XX:-UseGCOverheadLimit RunBB.jar
```

Start Program...
DROP SCHEMA IF EXISTS unielwin_setup
create schema unielwin_setup
USE unielwin_setup

Building Time(ms): 720 ms.

Build CT_RChain_TABLES for length = 2 are DONE
Rchain! are done

delete from `a,b_CT` where MULT='0';
Building CT Time(ms): 1814 ms.
The CT database is ready for use.

CSVPrecomputor TOTAL Time(ms): 32 ms.
CSV files are generated.

Structure Learning is DONE. Now doing the parameter learning.
rchain: `a,b`

Parameter Learning Time(ms): 2445 ms.
Parameter learning is done.
Start Writing MLN...

create table `a_CP_smoothed` as select * from `a_CP`;
create table `b_CP_smoothed` as select * from `b_CP`;
create table `rating(course0)_CP_smoothed` as select * from `rating(course0)_CP`;

BIF Generator starts
BIF Generator Ends for unielwin
smoothed_CP Time(ms): 2129 ms.

Finish running BayesBaseH.
Total Running time is 11966ms.
11.2 Baseline test environment

For the baseline tests we did use the following hardware setup:

Processor: 3.3Ghz Quad core Intel Core i5
Haswell 4590
L1 cache:32k/32k L2 cache:256k(x4) L3 cache:6Mb
automatic Turbo boost 2.0 up to 3.7Ghz

Memory: 24G 1600 Mhz DDR3

Disk: 1 Tb Fushion Drive [ 128Gig of flash-storage with 1 TB harddrive ]

The software setup used for running the tests:

OSX: 15.5.0 Darwin Kernel Version 15.5.0: Tue Apr 19 18:36:36 PDT 2016;
root:xnu-3248.50.21~8/RELEASE_X86_64 x86_64

MySQL: 5.6.29 MySQL Community Server (GPL)
With the following MySQL settings altered from the defaults:
   innodb_buffer_size=4G
   join_buffer_size = 256M
   sort_buffer_size = 4M
   read_rnd_buffer_size = 4M

MySQL command line client:
   mysql Ver 14.14 Distrib 5.6.29, for osx10.8 (x86_64) using EditLine wrapper

MySQL Workbench:
   Version 6.3.6, build 511 (CE) 64bits.

Java: java version "1.8.0_91"
Java(TM) SE Runtime Environment (build 1.8.0_91-b14)
Java HotSpot(TM) 64-Bit Server VM (build 25.91-b14, mixed mode)

FactorBase: RunBB.jar
Available at: http://www.cs.sfu.ca/~oschulte/BayesBase/download.html
11.3 Baseline results FactorBase per dataset

11.3.1 Financial std dataset

![Figure 1A](image1.png)

![Figure 1B](image2.png)
Exploiting relational database technology for statistical machine learning in FactorBase

Dataset info:

<table>
<thead>
<tr>
<th>Table</th>
<th>Number of Rows</th>
<th>Number of Columns</th>
<th>Size of data/index in DB</th>
</tr>
</thead>
<tbody>
<tr>
<td>acc</td>
<td>409</td>
<td>2</td>
<td>16KB</td>
</tr>
<tr>
<td>loan</td>
<td>409</td>
<td>5</td>
<td>48KB</td>
</tr>
<tr>
<td>Loan_Acc</td>
<td>409</td>
<td>2</td>
<td>48KB</td>
</tr>
<tr>
<td>Loan_Order</td>
<td>80220</td>
<td>2</td>
<td>6.5MB</td>
</tr>
<tr>
<td>Loan_Trans</td>
<td>894</td>
<td>2</td>
<td>96KB</td>
</tr>
<tr>
<td>order2</td>
<td>6471</td>
<td>4</td>
<td>304KB</td>
</tr>
<tr>
<td>trans</td>
<td>138123</td>
<td>5</td>
<td>7.5MB</td>
</tr>
</tbody>
</table>

Figure 1
Exploiting relational database technology for statistical machine learning in FactorBase

Financial_std EER:

Figure 1D
11.3.2 Hepatitis_std dataset

![Hepatitis_std with std deviation](image1.png)

![Hepatitis_std](image2.png)

Dataset info:

<table>
<thead>
<tr>
<th>Table</th>
<th>Number of Rows</th>
<th>Number of Columns</th>
<th>Size of data/index in DB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bio</td>
<td>32</td>
<td>3</td>
<td>48KB</td>
</tr>
<tr>
<td>dispat</td>
<td>500</td>
<td>4</td>
<td>64KB</td>
</tr>
<tr>
<td>indis</td>
<td>5691</td>
<td>11</td>
<td>1.6MB</td>
</tr>
<tr>
<td>inf</td>
<td>196</td>
<td>2</td>
<td>32KB</td>
</tr>
<tr>
<td>rel11</td>
<td>621</td>
<td>2</td>
<td>64KB</td>
</tr>
<tr>
<td>rel12</td>
<td>5691</td>
<td>2</td>
<td>304KB</td>
</tr>
<tr>
<td>rel13</td>
<td>196</td>
<td>2</td>
<td>32KB</td>
</tr>
</tbody>
</table>
Figure 1G

Hepatitis_std EER:

Figure 1H
11.3.3 MovieLens_std dataset

Dataset info:

<table>
<thead>
<tr>
<th>Table</th>
<th>Number of Rows</th>
<th>Number of Columns</th>
<th>Size of data/index in DB</th>
</tr>
</thead>
<tbody>
<tr>
<td>item2</td>
<td>1682</td>
<td>4</td>
<td>224KB</td>
</tr>
<tr>
<td>u2base</td>
<td>79710</td>
<td>3</td>
<td>12.1MB</td>
</tr>
<tr>
<td>user</td>
<td>941</td>
<td>3</td>
<td>96KB</td>
</tr>
</tbody>
</table>

Figure 1L

Figure 1J

Figure 1K
MovieLens_std EER:

Figure 1L
11.3.4 MovieLens_TQ dataset

**Dataset info:**

<table>
<thead>
<tr>
<th>Table</th>
<th>Number of Rows</th>
<th>Number of Columns</th>
<th>Size of data/index in DB</th>
</tr>
</thead>
<tbody>
<tr>
<td>item2</td>
<td>3883</td>
<td>4</td>
<td>144KB</td>
</tr>
<tr>
<td>u2base</td>
<td>998842</td>
<td>3</td>
<td>83.2MB</td>
</tr>
<tr>
<td>user</td>
<td>6201</td>
<td>3</td>
<td>496KB</td>
</tr>
</tbody>
</table>

**Figure 1M**

**Figure 1N**
Exploiting relational database technology for statistical machine learning in FactorBase

MovieLens_TQ EER:
11.3.5 Mutagenesis_std dataset

Dataset info:

<table>
<thead>
<tr>
<th>Table</th>
<th>Number of Rows</th>
<th>Number of Columns</th>
<th>Size of data/index in DB</th>
</tr>
</thead>
<tbody>
<tr>
<td>bond</td>
<td>4657</td>
<td>3</td>
<td>624KB</td>
</tr>
<tr>
<td>Mole</td>
<td>185</td>
<td>6</td>
<td>96KB</td>
</tr>
<tr>
<td>moleatm</td>
<td>4805</td>
<td>2</td>
<td>304KB</td>
</tr>
</tbody>
</table>
Exploiting relational database technology for statistical machine learning in FactorBase

Mutagenesis std EER:

Figure 1T
11.3.6 Unielwin dataset

Dataset info:

<table>
<thead>
<tr>
<th>Table</th>
<th>Number of Rows</th>
<th>Number of Columns</th>
<th>Size of data/index in DB</th>
</tr>
</thead>
<tbody>
<tr>
<td>course</td>
<td>10</td>
<td>3</td>
<td>48KB</td>
</tr>
<tr>
<td>prof</td>
<td>6</td>
<td>3</td>
<td>48KB</td>
</tr>
<tr>
<td>RA</td>
<td>25</td>
<td>4</td>
<td>80KB</td>
</tr>
<tr>
<td>registration</td>
<td>92</td>
<td>4</td>
<td>80KB</td>
</tr>
<tr>
<td>student</td>
<td>38</td>
<td>3</td>
<td>48KB</td>
</tr>
</tbody>
</table>
Exploiting relational database technology for statistical machine learning in FactorBase

unielwin EER:

Figure 1X
11.4 Dynamic view testing results

The dynamic view testing results are acquired using the same hardware that is used for the baseline benchmark. The data-sets are identical and the database software is the same MySQL version. In the subsequent paragraphs we list the results per data-set using a diagram and we repeat the data-set information just for clarity. In each diagram we display the results for the CT building time and parameter learning step in milliseconds.

11.4.1 Financial_std dataset

![Diagram showing CT building time and parameter learning for Financial_std dataset with data points at 1845 and 3130 milliseconds respectively.]

**Dataset info:**

<table>
<thead>
<tr>
<th>Table</th>
<th>Number of Rows</th>
<th>Number of Columns</th>
<th>Size of data/index in DB</th>
</tr>
</thead>
<tbody>
<tr>
<td>acc</td>
<td>409</td>
<td>2</td>
<td>16KB</td>
</tr>
<tr>
<td>loan</td>
<td>409</td>
<td>5</td>
<td>48KB</td>
</tr>
<tr>
<td>Loan_Acc</td>
<td>409</td>
<td>2</td>
<td>48KB</td>
</tr>
<tr>
<td>Loan_Order</td>
<td>80220</td>
<td>2</td>
<td>6.5MB</td>
</tr>
<tr>
<td>Loan_Trans</td>
<td>894</td>
<td>2</td>
<td>96KB</td>
</tr>
<tr>
<td>order2</td>
<td>6471</td>
<td>4</td>
<td>304KB</td>
</tr>
<tr>
<td>trans</td>
<td>138123</td>
<td>5</td>
<td>7.5MB</td>
</tr>
</tbody>
</table>

![Table showing dataset information for Financial_std with columns for Table, Number of Rows, Number of Columns, and Size of data/index in DB.]

101
11.4.2 Hepatitis_std dataset

Dataset info:

<table>
<thead>
<tr>
<th>Table</th>
<th>Number of Rows</th>
<th>Number of Columns</th>
<th>Size of data/index in DB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bio</td>
<td>32</td>
<td>3</td>
<td>48KB</td>
</tr>
<tr>
<td>dispat</td>
<td>500</td>
<td>4</td>
<td>64KB</td>
</tr>
<tr>
<td>indic</td>
<td>5691</td>
<td>11</td>
<td>1.6MB</td>
</tr>
<tr>
<td>inf</td>
<td>196</td>
<td>2</td>
<td>32KB</td>
</tr>
<tr>
<td>rel11</td>
<td>621</td>
<td>2</td>
<td>64KB</td>
</tr>
<tr>
<td>rel12</td>
<td>5691</td>
<td>2</td>
<td>304KB</td>
</tr>
<tr>
<td>rel13</td>
<td>196</td>
<td>2</td>
<td>32KB</td>
</tr>
</tbody>
</table>
11.4.3 MovieLens_std dataset

Dataset info:

<table>
<thead>
<tr>
<th>Table</th>
<th>Number of Rows</th>
<th>Number of Columns</th>
<th>Size of data/index in DB</th>
</tr>
</thead>
<tbody>
<tr>
<td>item2</td>
<td>1682</td>
<td>4</td>
<td>224KB</td>
</tr>
<tr>
<td>u2base</td>
<td>79710</td>
<td>3</td>
<td>12.1MB</td>
</tr>
<tr>
<td>user</td>
<td>941</td>
<td>3</td>
<td>96KB</td>
</tr>
</tbody>
</table>

Figure MovieLens_std result 1

Figure MovieLens_std result 2
11.4.4 MovieLens_TQ dataset

Dataset info:

<table>
<thead>
<tr>
<th>Table</th>
<th>Number of Rows</th>
<th>Number of Columns</th>
<th>Size of data/index in DB</th>
</tr>
</thead>
<tbody>
<tr>
<td>item2</td>
<td>3883</td>
<td>4</td>
<td>144KB</td>
</tr>
<tr>
<td>u2base</td>
<td>998842</td>
<td>3</td>
<td>83.2MB</td>
</tr>
<tr>
<td>user</td>
<td>6201</td>
<td>3</td>
<td>496KB</td>
</tr>
</tbody>
</table>
11.4.5 Mutagenesis_std dataset

**Figure Mutagenesis_std result 1**

**Dataset info:**

<table>
<thead>
<tr>
<th>Table</th>
<th>Number of Rows</th>
<th>Number of Columns</th>
<th>Size of data/index in DB</th>
</tr>
</thead>
<tbody>
<tr>
<td>bond</td>
<td>4657</td>
<td>3</td>
<td>624KB</td>
</tr>
<tr>
<td>Mole</td>
<td>185</td>
<td>6</td>
<td>96KB</td>
</tr>
<tr>
<td>moleatm</td>
<td>4805</td>
<td>2</td>
<td>304KB</td>
</tr>
</tbody>
</table>

**Figure Mutagenesis_std result 2**
11.4.6 Unielwin dataset

Dataset info:

<table>
<thead>
<tr>
<th>Table</th>
<th>Number of Rows</th>
<th>Number of Columns</th>
<th>Size of data/index in DB</th>
</tr>
</thead>
<tbody>
<tr>
<td>course</td>
<td>10</td>
<td>3</td>
<td>48KB</td>
</tr>
<tr>
<td>prof</td>
<td>6</td>
<td>3</td>
<td>48KB</td>
</tr>
<tr>
<td>RA</td>
<td>25</td>
<td>4</td>
<td>80KB</td>
</tr>
<tr>
<td>registration</td>
<td>92</td>
<td>4</td>
<td>80KB</td>
</tr>
<tr>
<td>student</td>
<td>38</td>
<td>3</td>
<td>48KB</td>
</tr>
</tbody>
</table>

Figure unielwin result 1

Figure unielwin result 2
11.5 Source code samples

Below we list a partial piece of the SQL used to dynamically generate the SQL for the creation of views. The purpose of this listing is to get an idea on how prepared statements are generated. It is not a full listing of all code used to implement the various algorithms.

Sample 1: Main stored procedure that can be executed from the client program:

```sql
CREATE PROCEDURE LearnDataModel(pdb VARCHAR(255)) BEGIN
DECLARE childtablename VARCHAR(255);
DECLARE parenttablename1 VARCHAR(255);
DECLARE parenttablename2 VARCHAR(255);
DECLARE parenttablename3 VARCHAR(255);
DECLARE parenttablename4 VARCHAR(255);
DECLARE loop_cntr INT DEFAULT 0;
DECLARE num_rows INT DEFAULT 0;
DECLARE num_relations INT DEFAULT 0;
BEGIN
/* Create the count tables for this database */
call LearnDataModel_0(pdb);
DROP TEMPORARY TABLE IF EXISTS relations;
CREATE TEMPORARY TABLE relations(name varchar(255));
BEGIN
/* Start determining the relations by selecting the child tables. */
DECLARE ChildTables_cursor CURSOR FOR
SELECT DISTINCT u.table_name as 'ChildTable'
FROM information_schema.table_constraints AS c
JOIN information_schema.key_column_usage AS u
USING(constraint_schema,constraint_name)
WHERE c.constraint_type = 'FOREIGN KEY'
AND u.referenced_table_schema = pdb
GROUP BY u.table_name;
/* Declare exception handler */
DECLARE CONTINUE HANDLER FOR NOT FOUND
SET no_more_rows = TRUE;
OPEN ChildTables_cursor;
select FOUND_ROWS() into num_rows;
the_loop: LOOP
FETCH ChildTables_cursor INTO childtablename;
/* Leave loop if not records found or all rows are processed */
IF no_more_rows THEN
CLOSE ChildTables_cursor;
LEAVE the_loop;
END IF;
/*
** Build a temporary table with the info we need. */
DROP TEMPORARY TABLE IF EXISTS parenttables;
DROP TEMPORARY TABLE IF EXISTS alltables;
DROP TEMPORARY TABLE IF EXISTS joinclause;
CREATE TEMPORARY TABLE parenttables
SELECT u.referenced_table_name
FROM information_schema.table_constraints AS c
JOIN information_schema.key_column_usage AS u
USING(constraint_schema,constraint_name)
WHERE c.constraint_type = 'FOREIGN KEY'
AND c.table_schema = pdb
AND u.table_name = childtablename
ORDER BY u.table_schema,u.table_name,u.column_name;
END;```
CREATE TEMPORARY TABLE alltables
SELECT table_name, column_name, column_key
FROM INFORMATION_SCHEMA.COLUMNS
WHERE table_name=childtablename
AND table_schema = pdb
UNION ALL
SELECT table_name, column_name, column_key
FROM INFORMATION_SCHEMA.COLUMNS
WHERE TABLE_NAME IN (SELECT referenced_table_name FROM parenttables)
AND table_schema = pdb;
/* Generate the relation name */
SET @RELATIONNAME = (SELECT GROUP_CONCAT(DISTINCT table_name separator '_')
FROM alltables);
/* Extract the column data to create the join clauses */
CREATE TEMPORARY TABLE joinclause
SELECT table_name, column_name, referenced_table_name, referenced_column_name,
constraint_name
FROM information_schema.key_column_usage
WHERE (KEY_COLUMN_USAGE.TABLE_SCHEMA = pdb)
AND table_name = childtablename
AND constraint_name != "PRIMARY"
ORDER BY TABLE_NAME;
/* Create the WHERE clause, use GROUP_CONCAT() to concatenate
** multiple rows if needed. Use the separator 'AND' to
** connect multiple join-clauses if there are multiple clauses.
*/
SET @WHERECLAUSE = (SELECT GROUP_CONCAT(table_name, '.', column_name, " = ",
referenced_table_name, '.', referenced_column_name
SEPARATOR ' AND ')
FROM joinclause);
/* Drop view first, if exists. */
SET @SQLQUERY = 'DROP VIEW IF EXISTS ';
PREPARE stmt1 FROM @SQLQUERY;
EXECUTE stmt1;
DEALLOCATE PREPARE stmt1;
SET @SQLQUERY = NULL;
SELECT CONCAT("Creating view ", @RELATIONNAME) AS 'View';
SET @SQLQUERY = CONCAT("CREATE VIEW ", @RELATIONNAME, " AS ");
/* The SELECT <column list> */
SET @SQLQUERY = CONCAT(@SQLQUERY, " AS MULT, ");
SET @COLLIST = (SELECT GROUP_CONCAT(CONCAT(table_name, '.', column_name))
FROM alltables
WHERE column_key != "PRI");
/* The table FROM list */
SET @SQLQUERY = CONCAT(@SQLQUERY, FROM alltables);
/* The SELECT <column list> */
SET @SQLQUERY = CONCAT(@SQLQUERY, " FROM ", @COLLIST);
SET @SQLQUERY = CONCAT(@SQLQUERY, " WHERE ", @WHERECLAUSE);
SET @SQLQUERY = CONCAT(@SQLQUERY, " GROUP BY ");
SET @SQLQUERY = CONCAT(@SQLQUERY, @RELATIONNAME);
SET @SQLQUERY = CONCAT(@SQLQUERY, " ORDER BY MULT ");
PREPARE stmt1 FROM @SQLQUERY;
EXECUTE stmt1;
DEALLOCATE PREPARE stmt1;
SET @SQLQUERY = NULL;
/* Generate the 'flat' and 'start' views which later on will
** be sort/merged to create the 'false' table.
** The flat table is a SELECT with the aggregate SUM() on
** the columns that exist in the entity tables, so we exclude
** the connecting relation table.

```sql
SET @SQLQUERY = 'DROP VIEW IF EXISTS ';
SET @SQLQUERY = CONCAT('@SQLQUERY', ' ', @RELATIONNAME, '_flat');
PREPARE stmt1 FROM @SQLQUERY;
EXECUTE stmt1;
DEALLOCATE PREPARE stmt1;
SET @SQLQUERY = NULL;
SELECT CONCAT("Creating view ", @RELATIONNAME, '_flat') AS 'View';
```

```sql
SET @SQLQUERY = CONCAT('CREATE VIEW ', @RELATIONNAME, '_flat', ' AS ');
/* The SELECT <column list> */
SET @SQLQUERY = CONCAT(@SQLQUERY, 'SELECT SUM(MULT) AS MULT, ');
SELECT @COLLIST = (SELECT GROUP_CONCAT(column_name)
                   FROM INFORMATION_SCHEMA.COLUMNS
                   WHERE TABLE_NAME IN (SELECT referenced_table_name FROM parenttables)
                   AND table_schema = pdb
                   AND column_key != "PRI");
SET @SQLQUERY = CONCAT(@SQLQUERY, @COLLIST);
SET @SQLQUERY = CONCAT(@SQLQUERY, ' FROM ', @RELATIONNAME);
SET @SQLQUERY = CONCAT(@SQLQUERY, ' GROUP BY ');
SET @SQLQUERY = CONCAT(@SQLQUERY, @COLLIST);
SET @SQLQUERY = CONCAT(@SQLQUERY, ' ORDER BY MULT');
PREPARE stmt1 FROM @SQLQUERY;
EXECUTE stmt1;
DEALLOCATE PREPARE stmt1;
SET @SQLQUERY = NULL;
```

** The star table is a join over the count tables for the entity tables, multiplying both MULT columns to get the maximum cover.

```sql
SET @SQLQUERY = 'DROP VIEW IF EXISTS ';
SET @SQLQUERY = CONCAT('@SQLQUERY', ' ', @RELATIONNAME, '_star');
PREPARE stmt1 FROM @SQLQUERY;
EXECUTE stmt1;
DEALLOCATE PREPARE stmt1;
SET @SQLQUERY = NULL;
SELECT CONCAT("Creating view ", @RELATIONNAME, '_star') AS 'View';
```

```sql
SET @SQLQUERY = CONCAT('CREATE VIEW ', @RELATIONNAME, '_star', ' AS ');
/* The SELECT <column list> */
SET @SQLQUERY = CONCAT(@SQLQUERY, 'SELECT ');
SELECT @COLLIST = (SELECT GROUP_CONCAT( distinct table_name, '_counts', '.MULT' SEPARATOR '*')
                   FROM INFORMATION_SCHEMA.COLUMNS
                   WHERE TABLE_NAME IN (SELECT referenced_table_name FROM parenttables)
                   AND table_schema = pdb);
SET @COLLIST = CONCAT(@COLLIST, ' AS MULT, ');
SET @COLLIST = CONCAT(@COLLIST, (SELECT GROUP_CONCAT(column_name)
                   FROM INFORMATION_SCHEMA.COLUMNS
                   WHERE TABLE_NAME IN (SELECT referenced_table_name FROM parenttables)
                   AND table_schema = pdb
                   AND column_key != "PRI"));
SET @FROMLIST = (SELECT GROUP_CONCAT(distinct table_name, '_counts')
                   FROM INFORMATION_SCHEMA.COLUMNS
                   WHERE TABLE_NAME IN (SELECT referenced_table_name FROM parenttables)
                   AND table_schema = pdb);
SET @SQLQUERY = CONCAT('@SQLQUERY', ' ', @RELATIONNAME, '_false0');
PREPARE stmt1 FROM @SQLQUERY;
EXECUTE stmt1;
DEALLOCATE PREPARE stmt1;
SET @SQLQUERY = NULL;
SELECT CONCAT("Creating view ", @RELATIONNAME, '_false0') AS 'View';
```
Exploiting relational database technology for statistical machine learning in FactorBase

SET @SQLQUERY = CONCAT("CREATE VIEW ", @RELATIONNAME,"_false0", " AS "); /* The SELECT <column list> */
SET @SQLQUERY = CONCAT(@SQLQUERY, "SELECT IFNULL((s.MULT - f.MULT),s.MULT) AS MULT, ");

SET @COLLIST = (SELECT GROUP_CONCAT(CONCAT("s.", column_name)) FROM INFORMATION_SCHEMA.COLUMNS
WHERE TABLE_NAME IN (SELECT referenced_table_name FROM parenttables)
AND table_schema = pdb
AND column_key = "PRI");
SET @SQLQUERY = CONCAT(@SQLQUERY, @COLLIST);
SET @SQLQUERY = CONCAT(@SQLQUERY, " FROM ", @RELATIONNAME, ",_star s");
SET @JOIN = CONCAT(" LEFT JOIN ", @RELATIONNAME, ",_flat f ON ");
SET @WHERECLAUSE = (SELECT GROUP_CONCAT(CONCAT("s.", column_name, " = ", "f.", column_name)
SEPARATOR " AND ") FROM INFORMATION_SCHEMA.COLUMNS
WHERE TABLE_NAME IN (SELECT referenced_table_name FROM parenttables)
AND table_schema = pdb
AND column_key = "PRI");
SET @SQLQUERY = CONCAT(@SQLQUERY, @JOIN);
SET @WHERECLAUSE = (SELECT GROUP_CONCAT(CONCAT("s.", column_name, " = ", "f.", column_name)
SEPARATOR " AND ") FROM INFORMATION_SCHEMA.COLUMNS
WHERE TABLE_NAME IN (SELECT referenced_table_name FROM parenttables)
AND table_schema = pdb
AND column_key = "PRI");
SET @SQLQUERY = CONCAT(@SQLQUERY, @WHERECLAUSE);
PREPARE stmt1 FROM @SQLQUERY;
EXECUTE stmt1;
DEALLOCATE PREPARE stmt1;
SET @SQLQUERY = NULL;

SET @SQLQUERY = NULL;

SELECT CONCAT("Creating view ", @RELATIONNAME, ",false") AS "View";
SET @SQLQUERY = CONCAT("CREATE VIEW ", @RELATIONNAME,"_false", " AS ");
SET @SQLQUERY = CONCAT(@SQLQUERY, "SELECT * FROM ", @RELATIONNAME, ",_false0 ");
PREPARE stmt1 FROM @SQLQUERY;
EXECUTE stmt1;
DEALLOCATE PREPARE stmt1;
SET @SQLQUERY = NULL;

/* Create the final view and filter out the entries with 0 counts */
SELECT CONCAT("Creating view ", @RELATIONNAME, ",_false0") AS "View";
SET @SQLQUERY = CONCAT("CREATE VIEW ", @RELATIONNAME,"_false0", " AS ");
SET @SQLQUERY = CONCAT(@SQLQUERY, "WHERE MULT > 0 ");
PREPARE stmt1 FROM @SQLQUERY;
EXECUTE stmt1;
DEALLOCATE PREPARE stmt1;
SET @SQLQUERY = NULL;

SET @SQLQUERY = 'DROP VIEW IF EXISTS ';
SET @SQLQUERY = CONCAT(@SQLQUERY, @RELATIONNAME, ",CT");
PREPARE stmt1 FROM @SQLQUERY;
EXECUTE stmt1;
DEALLOCATE PREPARE stmt1;
SET @SQLQUERY = NULL;

SELECT CONCAT("Creating view ", @RELATIONNAME, ",CT") AS "View";
SET @SQLQUERY = CONCAT("CREATE VIEW ", @RELATIONNAME,"_CT", " AS SELECT "); /* The SELECT <column list> */
SET @COLLIST = (SELECT GROUP_CONCAT(distinct column_name) FROM INFORMATION_SCHEMA.COLUMNS
WHERE table_name = @RELATIONNAME
AND table_schema = pdb)
SET @SQLQUERY = CONCAT(@SQLQUERY, @COLLIST, ", 'T' AS RELATION", ", FROM ", @RELATIONNAME, ",_false0 ", tmp_join_, childtablename);
PREPARE stmt1 FROM @SQLQUERY;
EXECUTE stmt1;
DEALLOCATE PREPARE stmt1;
SET @SQLQUERY = NULL;

SET @SQLQUERY = NULL;

/* Count number of loops */
SET loop_cntr = loop_cntr + 1;
SET num_relations = num_relations + 1;

/* Save relation name */
INSERT INTO relations values(#RELATIONNAME);

END LOOP the_loop;

END;

SELECT CONCAT("Relations: ", num_relations) AS 'Number of relations examined';

/*
* Built the CT table that represents all tables in this schema
* call LearnDataModel_2(pdb);
*
* IF num_relations > 1 THEN
* SELECT CONCAT("Create CT tables for the combined : ", num_relations, " number of relations") AS "Final Step";
* call LearnDataModel_3(pdb);
* END IF;
* /

/* Build the CPTs for all parameters/columns */
SELECT 'Start building CPTs for all parameters';
call LearnDataModel_5(pdb);

call LearnDataModel_6(pdb);

END;

DELIMITER ;
Sample 2: Stored procedure executed to generate CPT views:

```sql
/*
** Generate a CPT for each parameter (read column) for the
datamodel loaded.
** - determine all columns that are part of a relation
** - use the CT data to generate the CPT
*/
CREATE PROCEDURE LearnDataModel_6(pdb VARCHAR(255))
BEGIN

DECLARE no_more_rows BOOLEAN;
DECLARE loop_cntr INT DEFAULT 0;
DECLARE num_rows INT DEFAULT 0;
DECLARE c_name VARCHAR(255);
DECLARE c_parent VARCHAR(255);
DECLARE c_tablename VARCHAR(255);
BEGIN

DECLARE column_cursor CURSOR FOR
SELECT name, parent, tablename
FROM columns_with_parent
ORDER BY name;

/* Declare exception handler */
DECLARE CONTINUE HANDLER FOR NOT FOUND
SET no_more_rows = TRUE;
OPEN column_cursor;
SELECT FOUND_ROWS() INTO num_rows;
the_loop: LOOP
FETCH column_cursor INTO c_name, c_parent, c_tablename;
/* Leave loop if not records found or all rows are processed */
IF no_more_rows THEN
CLOSE column_cursor;
LEAVE the_loop;
END IF;
/* Drop the existing CPT view for this column */
SET @SQLQUERY = 'DROP VIEW IF EXISTS '
SET @SQLQUERY = CONCAT(@SQLQUERY, c_name, '_CP0');
PREPARE stmt1 FROM @SQLQUERY;
EXECUTE stmt1;
DEALLOCATE PREPARE stmt1;
SET @SQLQUERY = NULL;

SET @SQLQUERY = CONCAT("CREATE VIEW ", c_name, "_CP0", " AS ");
SET @SQLQUERY = CONCAT(@SQLQUERY, "SELECT ", c_name, ", COUNT(*) AS local_mult, " );
SET @SQLQUERY = CONCAT(@SQLQUERY, "SELECT SUM(MULT) FROM ", c_parent, "_CT", WHERE ",
SET @SQLQUERY = CONCAT(@SQLQUERY, " " , c_name, " = " , c_tablename, " .", c_name, " ) AS parent_value FROM ",
SET @SQLQUERY = CONCAT(@SQLQUERY, " c_tablename, " GROUP BY ", c_name);
PREPARE stmt1 FROM @SQLQUERY;
EXECUTE stmt1;
DEALLOCATE PREPARE stmt1;
SET @SQLQUERY = NULL;

SET @SQLQUERY = CONCAT('DROP VIEW IF EXISTS ');
SET @SQLQUERY = CONCAT(@SQLQUERY, c_name, '_CP1');
PREPARE stmt1 FROM @SQLQUERY;
EXECUTE stmt1;
DEALLOCATE PREPARE stmt1;
SET @SQLQUERY = NULL;

SET @SQLQUERY = CONCAT("CREATE VIEW ", c_name, "_CP1", " AS ");
SET @SQLQUERY = CONCAT(@SQLQUERY, "SELECT ", c_name, ", 0 AS local_mult, " );
SET @SQLQUERY = CONCAT(@SQLQUERY, "SELECT SUM(MULT) FROM ", c_parent, "_CT", " WHERE ",
SET @SQLQUERY = CONCAT(@SQLQUERY, " = N/A " );
SET @SQLQUERY = CONCAT(@SQLQUERY, " c_tablename, " GROUP BY ", c_name);
PREPARE stmt1 FROM @SQLQUERY;
EXECUTE stmt1;
DEALLOCATE PREPARE stmt1;
```

112
Exploiting relational database technology for statistical machine learning in FactorBase

```
PREPARE stmt1 FROM @SQLQUERY;
EXECUTE stmt1;
DEALLOCATE PREPARE stmt1;
SET @SQLQUERY = NULL;

SET @SQLQUERY = 'DROP VIEW IF EXISTS ';
SET @SQLQUERY = CONCAT(@SQLQUERY, c_name, '_CP2');
PREPARE stmt1 FROM @SQLQUERY;
EXECUTE stmt1;
DEALLOCATE PREPARE stmt1;
SET @SQLQUERY = NULL;

SET @SQLQUERY = CONCAT("CREATE VIEW ", c_name, "_CP2", " AS ");
SET @SQLQUERY = CONCAT(@SQLQUERY, "SELECT ", c_name, ", local_mult/total AS CP, parent_value, local_mult, log(local_mult/total * local_mult) as likelihood", " FROM ", c_name, "_CP0 UNION SELECT ", c_name, ", 1.0/1.0 AS CP, parent_value, local_mult, 0.0 as likelihood FROM ", c_name, "_CP1" );
PREPARE stmt1 FROM @SQLQUERY;
EXECUTE stmt1;
DEALLOCATE PREPARE stmt1;
SET @SQLQUERY = NULL;

SET @SQLQUERY = CONCAT("CREATE VIEW ", c_name, "_CP", " AS SELECT * FROM ", c_name, "_CP2 WHERE ", c_name, " is not NULL" );
PREPARE stmt1 FROM @SQLQUERY;
EXECUTE stmt1;
DEALLOCATE PREPARE stmt1;
SET @SQLQUERY = NULL;

SELECT CONCAT("Creating CPT ", c_name, "_CP") as "CPT";
/* Count number of loops */
SET loop_cntr = loop_cntr + 1;
END LOOP the_loop;

END;
END|
DELIMITER ;
```