Learning Dependent-Concepts in ILP: Application to Model-Driven Data Warehouse

Moez Essaidi, Aomar Osmani, Céline Rouveirol

LIPN - UMR CNRS 7030, Université Paris-Nord,
99 Avenue Jean-Baptiste Clément, 93430 Villetaneuse, France.
{essaidi, osmani, rouveirol}@lipn.univ-paris13.fr

Abstract. This paper studies a new machine learning application with possibly a challenging benchmark for relational learning systems. We are interested in the automation of *model-driven data warehouse* using machine learning techniques. The main goal is to automatically derive the transformation rules to be applied in the model-driven process. This aims to reduce the contribution of transformations designer and thereby reducing the time and cost of development. We propose to express the model transformation problem as an *Inductive Logic Programming* one: existing project traces (or projects experiences) are used to define the background knowledge and examples. The *Aleph* ILP engine is used to learn best transformation rules. In our application, we need to deal with several dependent-concepts. Taking into account the work in *Predicate Invention*, *Layered Learning*, *Cascade Learning* and *Context Learning*, we propose a new methodology that automatically updates the background knowledge of concepts to be learned. Experimental results support the conclusion that our approach is suitable to solve this kind of problem.

1 Overview

The model-driven engineering [2, 3] is an approach that organizes the development of a system around the design of models (conform to metamodels) and the definition of transformations to generate required components. The *model-driven data warehouse* represents approaches [43, 21, 9] that apply the MDA[1] standard for the development of the data warehouse systems [41]. The approach presented in [43] describes derivation of *Online-Analytical-Processing (OLAP)* schemas from *Entity-Relationship (ER)* schemas. The source and target schemas are respectively conform to ER and OLAP metamodels of the *Common-Warehouse-Metamodel*. Authors describe how an ER schema is mapped to an OLAP schema and provide, also, a set of *Query-View-Transformation* rules *(e.g., EntityToCube, AttributeToMeasure, RelationShipToDimension)* to ensure this. The approach presented in [21] extends the *Unified-Modeling-Language* and the *Common-Warehouse-Metamodel* to multidimensional modeling with MDA. Authors focus

[1] The *Object Management Group (http://www.omg.org/mda)* proposes the *Model-Driven Architecture* as standard implementation of the model-driven engineering.

on the transformation of the multidimensional conceptual model (i.e., conceptual OLAP schema) to the multidimensional logical model (i.e., logical OLAP schema). They provide, using the *Query-View-Transformation* language, transformations *(e.g., Fact2Table, Dimension2Table, etc.)* to derive the logical schema from the conceptual one.

We provide in [9] a unified model-driven data warehouse approach including an integrated design framework and transformation process. We propose the *UML CORE* metamodel to design the operational source-model and the *CWM OLAP* metamodel to design the multidimensional target-model. However, model transformations definition requires serious skills within metamodels and transformation languages. In this context, the *model transformation by-example* approach [1, 35, 7, 14] proposes to create automatically model transformations from pairs of source and target model examples. Then, in [10], we extend our proposal by a conceptual transformation-by-example framework for the model-driven data warehouse context. For example, the $ClassToCube$ relation represents a mapping of $Class$, $Property$ and $RelationShip$ elements of the *UML CORE* into $Cube$, $Measure$ and $CubeDimensionAssocition$ elements of the *CWM OLAP*. The input-instances (i.e., a, p, rse) of $Class$, $Property$ and $Relationship$ define elememts of a candidate source-model; this input-pattern gives the context in the source-model when a class is transformed into cube. The output-instances (i.e., c, m, cda) of $Cube$, $Measure$ and $CubeDimensionAssocition$ define the generated elements of a target-model; this output-pattern states the context in the target-model where a cube is generated from a class.

This work extends the proposed method (in previous works) by machine learning in order to reduce expert contribution in the transformation process. We propose to express the models transformation problem as an *Inductive Logic Programming* [25, 20] one and to use existing projects trace to find the best transformation rules. To the best of our knowledge, this work is the only one effort that has been developed for automating *model-driven data warehouse* with relational learning and it is the first effort that provides real experimental results in this context. In the *model-driven data warehouse* application, we find dependencies between transformations. We investigate a new machine learning methodology stemming from the application needs: learning Dependent-Concept. Following work about *Layered Learning* [34, 27, 16], *Predicate Invention* [26, 31, 32], *Context Learning* [38, 4, 39] and *Cascade Learning* [13, 37, 42], we propose a *Dependent-Concept Learning (DCL)* approach where the objective is to build a pre-order set of concepts on this dependency relationship: first learn non dependent concepts, then at each step, the theories of learned concepts are added as background knowledge to the future concepts to be learned with the respect to this given pre-order, and so on. This DCL methodology is implemented and applied to our transformation learning problem using Aleph[2]. The experimental evaluation s that the DCL system gives significantly better results.

The remainder of the paper is structured as follows. Section 2 provides background definitions and presents the application domain. Section 3 details

[2] `http://www.cs.ox.ac.uk/activities/machlearn/Aleph/`

the used machine learning algorithms and introduces the *Dependent-Concept Learning* approach. Section 4 reports experimental results. Section 5 gives our conclusions and future work.

2 Background Definitions

Definition 1 (Model). A model $M = (G, MM, \mu)$ is a tuple where: $G = (N_G, E_G, \Gamma_G)$ is a directed multigraph [3], MM is itself a model called the reference model of M (i.e., its metamodel) associated to a graph $G_{MM} = (N_{MM}, E_{MM}, \Gamma_{MM})$, and $\mu : N_G \cup E_G \rightarrow N_{MM}$ is a function associating elements (nodes and edges) of G to nodes of G_{MM}.

The relation between a model and its reference model (metamodel) is called conformance and is noted $conformsTo$. Elements of MM are called metaelements (or meta-concepts). μ is neither injective (several model elements may be associated to the same metaelement) nor surjective (not all metaelements need to be associated to a model element) [18]. In the ILP framework (regarding the background knowledge and examples), a model M_i is characterized by its description MD_i, i.e., a set of predicates that correspond to the contained elements. The predicates used to represent M_i as logic programs are extracted from its metamodel ω_i. For example, consider a data model used to manage customers and invoices. The classes $Customer$ and $Invoice$ are defined respectively by $class(customer)$ and $class(invoice)$. The one-to-many association that relate $Customer$ to $Invoice$ is mainly defined by $association(customer - invoice, customer, invoice)$ (others predicates, presented next, are used to define multiplicitis of the association). Then, the logic description of models from project's traces constitutes the generated background knowledge program in ILP.

Definition 2 (Metamodel and Meta-Metamodel). A meta-metamodel is a model that is its own reference model (i.e., it conforms to itself). A metamodel is a model such that its reference model is a meta-metamodel [18]. The metamodeling architecture (part of the *model-driven-architecture* standard) is based on metalevels: $M3$, $M2$, $M1$ and $M0$. $M3$ is the meta-metamodel level and it forms the foundation of the metamodeling hierarchy (the *Meta-Object-Facility* is an example of meta-metamodel). $M2$ consists of the metamodel level and The *Unified-Modeling-Language* and the *Common-Warehouse-Metamodel* are examples of metamodels. $M1$ regroups all user-defined models and $M0$ reprensents the runtime instances of models.

The basic idea is to specify the relations among source and target element types using constraints. However, declarative constraints can be given executable semantics, such as in logic programming. In fact, logic programming with its unification-based matching, search, and backtracking seems a natural choice to implement the relational approach, where predicates can be used to describe the relations [5]. For example, in [15] authors explore the application of logic

[3] A directed multigraph $G = (N_G, E_G, \Gamma_G)$ consists of a finite set of nodes N_G, a finite set of edges E_G, and a function $\Gamma_G : E_G \rightarrow N_G \times N_G$ mapping edges to their source and target nodes [18].

programming. In particular *Mercury*, a typed dialect of *Prolog*, and *F-logic*, an object-oriented logic paradigm, to implement transformations. In [30] authors discuss a formalization of modeling and model transformation using a generic formalism, the *Diagrammatic Predicate Logic (DPL)*. The *DPL* [6, 29] is a graph-based specification format that takes its main ideas from both categorical and first-order logic, and adapts them to software engineering needs.

Definition 3 (Model Transformation). A model transformation consists of a set of transformation rules which are defined by input and output patterns (denoted by $\mathbb{P}$) in $M2$ level. Formally, a model transformation is associated to a relation $R(MM, MN) \subseteq \mathbb{P}(MM) \times \mathbb{P}(MN)$ defined between two metamodels which allows to obtain a target model N conforming to MN from a source model M that conforms to metamodel MM [33].

Definition 4 (Transformation Example). A transformation example (or trace model) $R(M, N) = \{r_1, \ldots, r_k\} \subseteq \mathbb{P}(M) \times \mathbb{P}(N)$ specifies how the elements of M and N are consistently related by R. A base of examples is a set of transformation examples. The transformation examples represents project's traces or they can be collected from different experts [19].

For instance, we are interested in the transformation of the *Data-Source PIM* (denoted DSPIM) to the *Multidimensional PIM* (denoted MDPIM). The DSPIM represents a conceptual view of a data-source repository and its $conformsTo$ the UML CORE metamodel (part of the *Unified-Modeling-Language*). The MDPIM represents a conceptual view of a target data warehouse repository and its $conformsTo$ the *CWM OLAP* metamodel (part of the *Common-Warehouse-Metamodel*). The (i) definitions and examples of DSPIM/MDPIM, (ii) their respective metamodels (*UML CORE and CWM OLAP*) and (iii) details about the proposed transformation-by-example framework for *Model-Driven Data Warehouse* are provided in our recent work [11]. The predicates extracted from the *UML CORE* metamodel to translate source models into logic program are: *type(name)*, *multiplicity(bound)*, *class(name)*, *property(name, type, lower, upper)*, *association(name, source, target)*, *associationOwnedAttribute(class, property)*, and *associationMemberEnds(association, property)*. Then, According to the *CWM OLAP* metamodel, the predicates defined to describe target models are: *cube(Name)*, *measure(Name, Type, Cube)*, *dimension(Name, isTime, isMeasure)*, *cubeDimensionAssociation(Cube, Dimension)*, *level(Name)*, *levelBasedHierarchy(Name, Dimension)*, and *hierarchyLevelAssociation(LevelBasedHierarchy, Level)*.

By analysing the source and target models, we observe that structural relationships (like aggregation relation, composition relation, semantic dependency, etc.) define a restrictive context for some transformations. For instance, let us consider the concept *PropertyToMeasure*. For instance, we know there is a composition relation between *Class* and *Property* and there is also a composition relation between *Cube* and *Measure* in the metamodels. This implies that the concept *PropertyToMeasure* must be considered only when the concept *ClassToCube* is learned. Therefore, the *ClassToCube* concept must be added as background knowledge in order to learn the *PropertyToMeasure* concept. This domain specificity induces a pre-order on the concept to be learned and defines a dependent-concept

learning problem. Therefore, in our approach, concepts are organized in order to defined a structure called *dependency graph*. In [8], Esposito et al. use the notion of dependency graph to deal with hierarchical theories. Authors define the dependency graph as a directed acyclic graph of concepts, in which parent nodes are assumed to be dependent on their offspring.

Definition 5 (Dependency Graph). A dependency graph is a directed acyclic graph of predicate letters, where an edge (p, q) indicates that atoms with predicate letter q are allowed to occur in the hypotheses defining the concept denoted by p [8].

3 Relational Learning of Dependent-Concept

The data warehouse is a database used for reporting; therefore a candidate language used to describe data is a relational database language. This language is close to *datalog* language used in relational learning (or *Inductive Logic Programming*). In addition, the conceptual models are defined in term of relations between elements of different types (properties, classes and associations). Therefore, it is natural to use supervised learning techniques handling concept languages with the same expressive level as manipulated data in order to exploit all information provided by the relationships between data. Even if there are quite a number of efficient machine learning algorithms that deal with attribute-value representations, relational languages allows encoding structural information fundamental for the transformation process. This is why ILP algorithms [25, 20] have been selected to deal with this learning problem. As ILP suffers from a scaling-up problem, the proposed architecture [9, 10] is designed in order to take into account this limitation. Thus, it's organised as a set of elementary transformations such that each one concerns a few number of predicates only, to reduce the search space. This section reminder the relational learning theory, introduces the *Dependent-Concept Learning* approach and compares it related concept-search approaches.

3.1 Relational Learning Setting

We consider the machine learning problem as defined in [22]. A (single) concept learning problem is defined as follows. Given i) a training set $E = E^+ \cup E^-$ of positive and negative examples drawn from an example language $\mathcal{L}_e$ ii) a hypothesis language $\mathcal{L}_h$, iii) optionally, some background knowledge B described in a relational language $\mathcal{L}_b$, iv) a generality relation $\geq$ relating formulas of $\mathcal{L}_e$ and $\mathcal{L}_h$, learning is defined as search in $\mathcal{L}_h$ for a hypothesis h such that h is consistent with E. A hypothesis h is consistent with a training set E if and only if it is both complete ($\forall e^+ \in E^+, h, B \geq e^+$) and correct ($\forall e^- \in E^-, h, B \not\geq e^-$). In an ILP setting, $\mathcal{L}_e$, $\mathcal{L}_b$ and $\mathcal{L}_h$ are Datalog languages, and most often, examples are ground facts or clauses, background knowledge is a set of ground facts or clauses and the generality relation is a restriction of deduction. As explained in [22], there are two main strategies for searching $\mathcal{L}_h$: either generate-and-test

or data-driven, and following any of those strategies, algorithms may proceed either top-down or bottom-up, or any combination of those. We used in our experiments the well known Aleph system, because of its ability to handle rich background knowledge, made of both facts and rules. Aleph follows a top-down generate-and-test approach.

It takes as input a set of examples, represented as a set of *Prolog* facts and background knowledge as a Datalog program. It also enables the user to express additional constraints C on the admissible hypotheses. Aleph tries to find a hypothesis $h \in \mathcal{L}_h$, such that h satisfying the constraints C and which is complete and partially correct. We used Aleph default mode: in this mode, Aleph uses a simple greedy set cover procedure and construct a theory H step by step, one clause at a time. To add a clause to the current target concept, Aleph selects an uncovered example as a seed, builds a most specific clause from this seed as the lowest bound of its search space and then performs an admissible search over the space of clauses that subsume this lower bound according the user clause length bound. In the next section, we show the reduction of the source-model, the target-model and the mapping between them as an ILP problem.

3.2 Dependent-Concept Learning Problem

Let $\{c_1, c_2, \ldots, c_n\}$ be a set of concepts to be learned in our problem. If we consider all the concepts independently, each concept c_i defines an independent ILP problem, i.e., all concepts have independent training sets E_i and share the same hypothesis language L_h and the same background knowledge B. We refer to this framework as the *Independent-Concept Learning (ICL)*. The second framework, *Dependent-Concept Learning (DCL)*, takes into account a pre-order relation[4] $\preceq$ between concepts to be learned such that $c_i \preceq c_j$ if the concept c_j depends on the concept c_i or in other term, if c_i is used to define c_j (Definition 5). More formally, a concept c_j is called *parent* of the concept c_i (or c_i is the *child or offspring* of c_j) if and only if $c_i \preceq c_j$ and there exists no concept c_k such that $c_i \preceq c_k \preceq c_j$. $c_i \preceq c_j$ denotes that c_j depends on c_i for its definition. A concept c_i is called *root* concept iff there exists no concept c_k such that $c_k \preceq c_i$ (in other words, a root concept c_i does not depend on any concept c_k, for $k \neq i$). The DCL framework uses the idea of decomposing a complex learning problem into a number of simpler ones. Then, it adapts this idea to the context of ILP multi-predicate learning.

A dependent-concept ILP learning algorithm is an algorithm that accepts a pre-ordered set of concepts, starts with learning root concepts, then child (or offspring) concepts and propagates the learned rules to the background knowledge of their parent concepts and continues recursively the learning process until all dependent-concepts have been learned. Within this approach, we benchmark two settings: (i) the background knowledge B_j of a dependent-concept (parent) c_j is extended with the child concept *instances* (as a set of facts – this framework is refered to as DCLI) and (ii) B_j is extended with child concept intensional

[4] A pre-order is a binary relationship reflexive and transitive.

definitions: all child concepts are learned as sets of rules and are added to B_j – this frameworks is refered to as DCLR in the following sections. In both cases, DCLI or DCLR, all predicates representing child of c_j can be used in the body of c_j's definition. Our claim here is that the quality of the c_j's theory substantially improves if all its child concepts are known in B_j, extensionnally or intensionnally. Section 4 provides results concerning the impact of child concepts' representation (extensional vs. intensional) on the the quality of the c_j. Finally, the task of empirical *Dependent-Concept Learning* in ILP, which is concerned with learning a set of concepts based on a dependency-graph and given a set of examples and background knowledge, can be formulated as follows:

Given: A dependency graph $G_d = (C_d, E_d)$ where $Cd = \{c_1, c_2, \ldots, c_n\}$ the set of concepts to learn such that $\forall c_i \in C_d$:

- A set of transformation examples (i.e., examples) $E = \{E_1, E_2, \ldots, E_n\}$ is given; and defined as (where $|TM|$ is the number of training models):

$$E_i = \{R_i^j(M^j, N^j) \mid R^j(M^j, N^j) \subseteq \mathbb{P}(M^j) \times \mathbb{P}(N^j),\ j \leq |TM|\}$$

- Background knowledge B which provide additional information about the examples and defined as:

$$B = \{\mathbb{P}(M^j)\ \cup\ \mathbb{P}(N^j) \mid M^j\ conformsTo\ MM,\ N^j\ conformsTo\ MN)\}$$

Find: $\forall c_i \in C_d$, based on E_d and following a *BFS* strategy[5], learn a transformation rule $R_i(MM, MN) \subseteq \mathbb{P}(MM) \times \mathbb{P}(MN)$; where MM is the reference source-metamodel and MN is the reference target-metamodel.

3.3 Comparison With Related Concept Search Problems

Stone et al. introduce in [34] the *Layered Learning* machine learning paradigm. In [27] authors study the problem of constructing the approximation of higher level concepts by composing the approximation of lower level concepts. Authors in [16, 17] present an alternative to standard genetic programming that applies layered learning techniques to decompose a problem. The layered learning approach presented by Muggleton in [24] aims at the construction of a large theory in small pieces. Compared to layered learning, the DCL approach aims to find all concepts theory using the theories of concepts on which they depend. Then, while the layered learning approach exploits a bottom-up, hierarchical task decomposition, the DCL algorithm exploits the dependency relationships between specific concepts of the given dependency graph. The dependency structure in [34] is a hierarchy, whereas our dependency structure is a *directed acyclic graph*. A breadth-first search algorithm is used to explore the dependency graph.

Within the field of *Inductive Logic Programming* the term *Predicate Invention* is introduced [23] and it involves the decomposition of predicates being learned

[5] Start by an offspring and non-dependent concept (i.e., a root concept), then follow its parents dependent-concepts

into useful sub-concepts. Muggleton in [26] defines *Predicate Invention* as the augmentation of a given theoretical vocabulary to allow finite axiomatisation of the observational predicates. In [31, 32], Stahl study the utility of predicate invention task in ILP and its capabilities as a bias shift operation. Rios et al. investigate in [28] on specification language extension when no examples are explicitly given of the invented predicate. The DCL and *Predicate Invention* approaches share the fact they correspond to the process of introducing new theoretical relationships. However, in the case of *Predicate Invention*, the approach is usually based on decomposition of the theory to learn on simple sub-theories and the DCL approach is based on the composition of a theory from the learned theories.

In [13], authors introduces the *Cascade Generalization* method. This approach is compared to other approaches that generate and combine different classifiers like the *Stacked Generalization* approach [40, 36, 37]. In [42], Xie proposes several speed-up variants of the original cascade generalization and show that the proposed variants are much faster than the original one. As the *Cascade Generalization*, the DCL approach extends the background knowledge at each level by the information on concepts of the sub-level (according to the dependency graph). But, within the proposed DCL, we use the same classifiers for all iterations. In our experiments, we report the results of the extension of the background knowledge by instances (first setting named DCLI) and the learned theory (second setting named DCLR).

The *model transformation by-example* approach aims to find contextual patterns in the source model that map contextual patterns in target model. This task is defined as *Context Analysis* in [39]. The machine learning approaches that exploit context to synthesize concepts are proposed in [38, 4]. In [38] author provides a precise formal definition of context and list four general strategies for exploiting contextual information. Authors in [4] introduce an enhanced architecture that enables contextual learning in the *Neurosolver* (a problem solving system). Nevertheless, the notion of context is different in DCL. In fact, in the DCL, contextual information is the result of the learning process (which will form the transformation rule); while within the *Contextual Learning* strategy the context is part of input information's that improve the performance of the learner.

4 Empirical Results

This section describes the experimental setup and compares the results of the two tested methods: The *Independent-Concept Learning (ICL)* and the *Dependent-Concept Learning (DCL)*.

4.1 Materials and Methods

For the experimentations presented in [11], we use a set of real-world data models provided by an industrial partner. Concerning the experimentations of this

paper, we use the *Microsoft AdventureWorks 2008R2* sample database family[6] reference databases. The *AdventureWorksOLTP* is a sample operational database used to define the source-model (i.e., the data-source schema – DSPIM). The *AdventureWorksDW* is a sample data warehouse schema used as target-model (i.e., the multidimensional schema – MDPIM). The *AdventureWorksOLTP*, *AdventureWorksDW* and the mapping between them (evaluated by the expert) is considered as a reference project-trace. This will allow us to benchmark our approach on a new extended schema (that generate more examples) and a new dependency graph. The databases elements (i.e., classes, properties and associations) are encoded as background knowledge (B) and the mapping instances between their elements allows to define positive (E^+) and negative (E^-) examples. Concerning the number of examples, we have $\|E_{ClassToCube}\| = 71$ denoting the number of example (positives and negatives) used to learn $ClassToCube$ concept, $\|E_{PropertyToMeasure}\| = 249$, $\|E_{PropertyToDimension}\| = 245$, $\|E_{RelationShipToDimension}\| = 93$, $\|E_{ElementToHierarchyPath}\| = 338$, and $\|E_{ElementToDimensionLevel}\| = 338$.

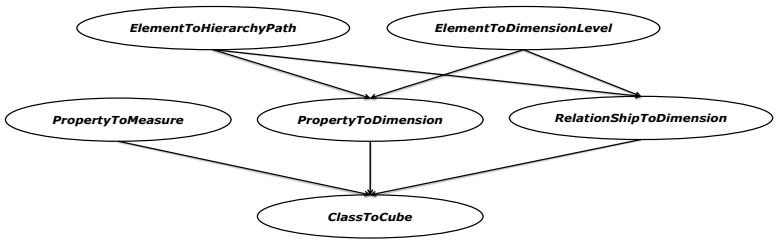

Fig. 1. The Considered Dependency Graph.

We use *Aleph* ILP engine, to learn first-order rules. We run *Aleph* in the default mode, except for the *minpos* and *noise* parameters: *:- set(minpos, p)* establishes as p the minimum number of positive examples covered by each rule in the theory (for all experiments we fix $p = 2$); and *:- set(noise, n)* is used to reports learning performance by varying the number of negative examples allowed to be covered by an acceptable clause (we use two setting $n = 5$ and $n = 10$). Then, a *Prolog* compiler is needed to run *Aleph*. We use *YAP (Yet Another Prolog)* [7], an optimized open-source *Prolog* platform. We propose to compare the following approaches:

1. The *Independent-Concept Learning (ICL)* approach, which proposes to learn the set of considered concepts independently.
2. The *Dependent-Concept Learning (DCL)* approach, which consider a dependency graph to learn the concepts. Within this approach, we benchmark two settings: (i) the background knowledge B of dependent-concepts (parent concepts) is updated with their child instances (denoted DCLI) and (ii) with

[6] `http://msftdbprodsamples.codeplex.com/`
[7] `http://www.dcc.fc.up.pt/~vsc/Yap/`

their chlid intensional definitions (denoted DCLR). We identify the concept dependencies illustrated by the graph in figure 1:

- *ClassToCube* $\preceq$ *PropertyToMeasure*: The *PropertyToMeasure* concept depends on the concept *ClassToCube*. In general, context of transforming *properties* depends on contextual information of transforming *classes* and the context of obtaining *measures* is part of context of obtaining *cubes*. In fact, *Properties* that become *Measures* are numeric *properties* of *classes* that become *cubes*. So, we need information about the context of *ClassToCube* transformation in order to find the context of *PropertyToMeasure*.
- *ClassToCube* $\preceq$ *PropertyToDimension*: This define dependency between *classes* transformed into *cubes* and their *properties* that can be transformed into *dimensions*. Regarding the *UML CORE* metamodel, we find a structural dependency between *Class* and *Property* elements (a *Class* includes attributes, represented by the *ownedAttribute* role that defines a set of *properties*). Then regarding the *CWM OLAP* metamodel, we have a structural dependency between *Cube* and *Dimension* elements. Current experiments confirm that structural dependencies in the metamodel act on the ways to perform learning.
- *ClassToCube* $\preceq$ *RelationShipToDimension*: Indeed, *dimensions* are, also, obtained from *relationships* of the *Class* that is transformed into *Cube*. The *CubeDimensionAssociation* meta-class relates a *Cube* to its defining dimensions as showed by the *CWM OLAP* metamodel in [11]. These relationships define the axes of analysis in the target multidimensional schema [41].
- *(PropertyToDimension, RelationShipToDimension)* $\preceq$ *ElementToHierarchyPath*: A *Dimension* has zero or more hierarchies. A *Hierarchy* is an organizational structure that describes a traversal pattern through a *Dimension*, based on parent/child relationships between members of a *Dimension*. Then, elements that are transformed into dimensions (*properties* and *relationships*) extend the background knowledge used to find hierarchy paths.
- *(PropertyToDimension, RelationShipToDimension)* $\preceq$ *ElementToDimensionLevel*: A *LevelBasedHierarchy* describes hierarchical relationships between specific levels of a *Dimension* (e.g., *Day, Month, Quarter and Year* levels for the *Time* dimension). So, rules of transforming elements into *Dimension* are used to find rules of obtaining the levels.

4.2 Results and Discussion

The first goal of this benchmark is to examine how the number of training models and examples influence the performances. Accuracy is commonly used for comparing the performances in machine learning and it is defined as $Accuracy = \frac{TP+TN}{P+N}$, where P (N) is the number of examples classified as positive (negative), TP (TN) is the number of examples classified as positive (negative) that are indeed positive (negative). In [11], we examine the accuracy of the learned rules to show the impact of the number of training models and examples and we report the obtained test accuracy curves for *ClassToCube* and *PropertyToDimension*. The

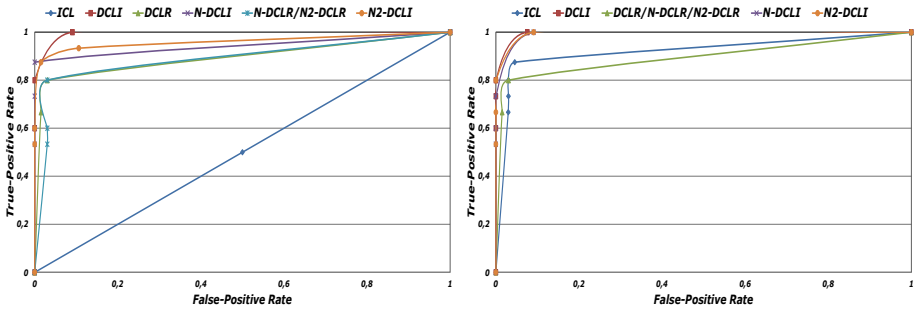

Fig. 2. *Learning PropertyToMeasure (n=5 for left) and (n=10 for right).*

accuracy of current experiments based on the new dataset (of *AdventureWorks*) confirm the results reported in [11].

The second goal of the analysis is to study the performances of the DCL approach (with the two settings DCLI and DCLR) compared to the ICL approach. The *Receiver-Operating-Characteristics (ROC)* graphs are a useful technique for visualizing, organizing and selecting classifiers based on their performance [12]. We report in this section the ROC curves of the tested approaches (ICL, DCLI and DCLR) based on the new dataset and the new enhanced dependency graph. The following metrics are used to report the ROC graphs: The $true-positive-rate$ (also called hit rate and recall = sensitivity) is estimated as $tp\ rate = \frac{TP}{P}$ and the $false-positive-rate$ (also called false alarm rate = 1 - specificity) as $fp\ rate = \frac{FP}{N}$. ROC graphs are two-dimensional graphs in which $tprate(sensitivity)$ is plotted on the Y axis and $fprate(1 - specificity)$ is plotted on the X axis. In order to assess the impact of a child concept rules quality on the learning performances of a parent concept, we experiment the case where the child concept is noisy. This experiment is made within the DCL approach, we add noise to the non-dependent concept (i.e., $ClassToCube$) and we observe results of learning dependent-concepts with different acceptable noise setting ($n = 5$ and $n = 10$). We report the cases where 10% (denoted N-DCLI and N-DCLR) and 20% (denoted N2-DCLI and N2-DCLR) of the examples are noisy (to add noise, we swap positives and negatives examples).

The area under the ROC curve, abbreviated AUC, is the common measure to compare the tested methods. The AUC represents, also, a measure of accuracy (results are reported by figures 2, 3, 4, 5 and 6). Figures show that $n = 10$ setting (right part of each figure) gives best performances compared to $n = 5$. This confirms that the choice of this parameter is important to deal with noisy information of database models in general. Comparing ICL, DCLI and DCLR approaches: results show that the DCLI has greater AUC than other tested methods. The DCLI curves follow almost the upper-left border of the ROC space. Therefore, it has better average performance compared to the DCLR and ICL ($AUC_{DCLI} > AUC_{DCLR} > AUC_{ICL}$). The ICL curves almost follow to the 45-degree diagonal of the ROC space, which represents a random classifier. The

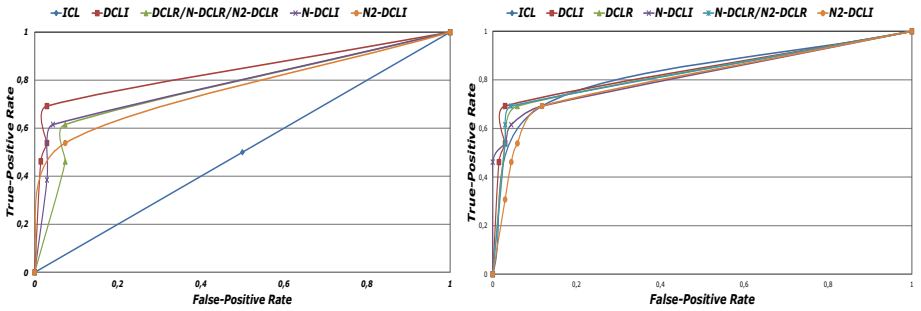

Fig. 3. *Learning PropertyToDimension (n=5 for left) and (n=10 for right).*

DCLR setting exhibits good results with respect to the ICL approach, which are nevertheless slightly worse than results of the DCLI setting.

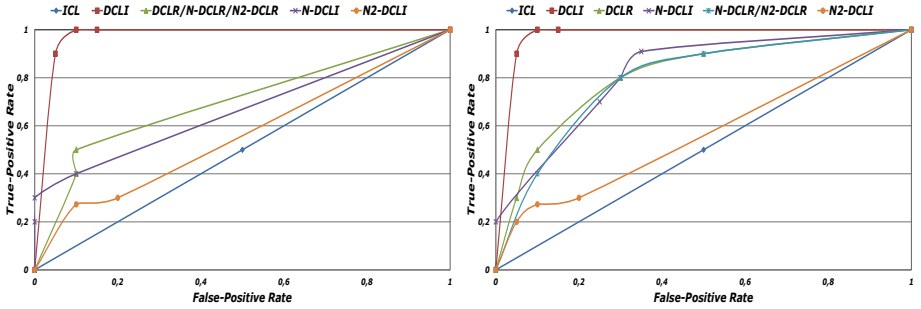

Fig. 4. *Learning RelationshipToDimension (n=5 for left) and (n=10 for right).*

The $AUC_{DCLI} > AUC_{DCLR} > AUC_{ICL}$ result is expected, because the DCLI configuration, when learning a parent concept, uses in its background knowledge offspring concepts as set of facts (extensional definition), as opposed to DCLR, which previously learns as sets of rules definition for offspring concepts. In case lower level concepts (i.e., offspring concepts) are not perfectly identified, the errors for offspring concepts propagate to parent concepts. We assume here that examples are noise-free, which explains why DCLI has a better behaviour than DCLR. Thus, for *PropertyToMeasure*, *PropertyToDimension* and *Relationship-ToDimension*, results integrate the error rate from *ClassToCube* learned rules. For the parent concepts *ElementToHierarchyPath* and *ElementToDimensionLevel* that depend on *(PropertyToDimension and RelationshipToDimension)*, results are influenced by the error rate propagation from learning *ClassToCube* and then *PropertyToDimension* and *RelationshipToDimension*.

Then, considering the N-DCLI/N2-DCLI and N-DCLR/N2-DCLR, we have mainly: $AUC_{N-DCLI} > AUC_{N2-DCLI}$ and $AUC_{N-DCLR} > AUC_{N2-DCLR}$. Curves show that the obtained performances depend on the concept to learn and its degree-of-dependence on *ClassToCube* (the noisy non-dependent concept of

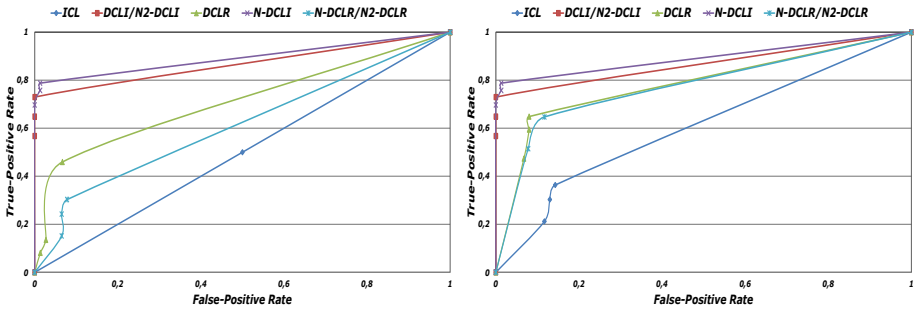

Fig. 5. *Learning ElementToHierarchyPath (n=5 for left) and (n=10 for right).*

this configuration). For instance, in figures 3 and **??**, *PropertyToDimension* and *RelationshipToDimension* are most impacted than *PropertyToMeasure* (in figure 2). The *PropertyToDimension* and *RelationshiptoDimension* concepts are highly dependent on *ClassToCube*. This can be observed on most schemas (remarks provided in [11]) and its confirmed by the expert point-of-view. For example, in the case of *RelationshipToDimension*, the N2-DCLI curve seems to reach the 45-degree diagonal. This gives us an idea on the noise that we can accept when learning specific dependency relationships. The *ElementToHierarchyPath* and *ElementToDimensionLevel* concepts are impacted by the noisy data of *ClassToCube*, but less than *PropertyToDimension* and *RelationshipToDimension*. We observe that *ElementToHierarchyPath* and *ElementToDimensionLevel* are not in direct dependence with *ClassToCube*.

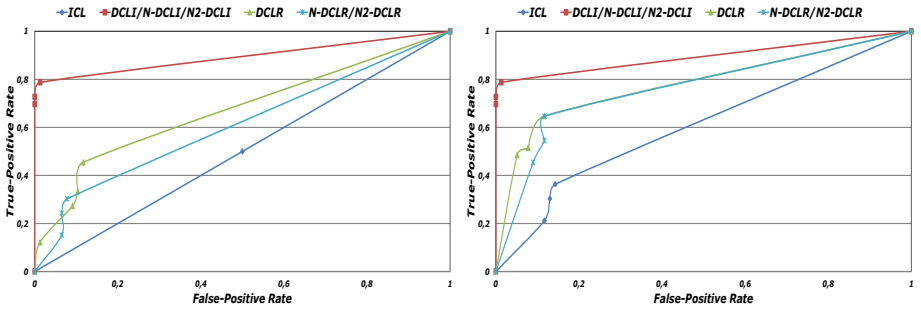

Fig. 6. *Learning ElementToDimensionLevel (n=5 for left) and (n=10 for right).*

5 Conclusion

This paper studies a real complex machine learning application: *model-driven data warehouse* automation using machine learning techniques. It includes the use of standard algorithms and a design of architecture to limit the impact of machine learning to the regions where learning from experience is needed. In addition, from

our application needs, we found an interesting machine learning problem: learning dependent-concept. Experimental results show that the proposed *Dependent-Concept Learning* approach to derive transformation rules in context of *model-driven data warehouse* gives significant performances improvement compared to the standard approach. From the business point-of-view, the learned theories are, in general, close to the ones given by human experts. Our future work will experiments the case when a business goals model is considered during transformations. For example, the derivation of the MDPIM from the pair (DSPIM, MDCIM), where MDCIM defines the organisation requirements/goals. We plan also to extend the approach to new application domains that provide a large dependency-graph (as example, the *Extraction, Transformation and Loading* process in the data warehousing architecture).

References

1. Z. Balogh and D. Varró. Model transformation by example using inductive logic programming. *Software and System Modeling*, 8(3):347–364, 2009.
2. J. Bézivin. On the unification power of models. *Software and System Modeling*, 4(2):171–188, 2005.
3. J. Bézivin. Model driven engineering: An emerging technical space. In *GTTSE*, pages 36–64. Springer, 2006.
4. A. Bieszczad and K. Bieszczad. Contextual learning in the neurosolver. In *ICANN*, pages 474–484. Springer, 2006.
5. K. Czarnecki and S. Helsen. Feature-based survey of model transformation approaches. *IBM Syst. J.*, 45:621–645, July 2006.
6. Z. Diskin and U. Wolter. A diagrammatic logic for object-oriented visual modeling. *Electr. Notes Theor. Comput. Sci.*, 203(6):19–41, 2008.
7. X. Dolques, M. Huchard, and C. Nebut. From transformation traces to transformation rules: Assisting model driven engineering approach with formal concept analysis. In *ICCS*, pages 093–106, Moscow, Russia, 2009.
8. F. Esposito, G. Semeraro, N. Fanizzi, and S. Ferilli. Multistrategy theory revision: Induction and abduction in inthelex. *Machine Learning*, 38(1-2):133–156, 2000.
9. M. Essaidi and A. Osmani. Model driven data warehouse using MDA and 2TUP. *Journal of Computational Methods in Sciences and Engineering*, 10:119–134, 2010.
10. M. Essaidi and A. Osmani. Towards Model-driven Data Warehouse Automation using Machine Learning. In *IJCCI (ICEC)*, pages 380–383, Valencia, Spain, 2010. SciTePress.
11. M. Essaidi, A. Osmani, and C. Rouveirol. Transformations learning in context of model-driven data warehouse: An experimental design based on inductive logic programming (accepted – to appear). In *ICTAI*. IEEE Computer Society, 2011.
12. T. Fawcett. Roc graphs: Notes and practical considerations for researchers. Technical report, HP Laboratories, 2004.
13. J. a. Gama and P. Brazdil. Cascade Generalization. *Mach. Learn.*, 41:315–343, December 2000.
14. I. García-Magariño, J. J. Gómez-Sanz, and R. Fuentes-Fernández. Model Transformation By-Example: An Algorithm for Generating Many-to-Many Transformation Rules in Several Model Transformation Languages. In *ICMT*, pages 52–66. Springer, 2009.
15. A. Gerber, M. Lawley, K. Raymond, J. Steel, and A. Wood. Transformation: The missing link of mda. In *ICGT*, pages 90–105. Springer, 2002.
16. S. M. Gustafson and W. H. Hsu. Layered Learning in Genetic Programming for a Cooperative Robot Soccer Problem. In *EuroGP*, pages 291–301, London, UK, 2001. Springer-Verlag.

17. D. Jackson and A. P. Gibbons. Layered learning in boolean GP problems. In *EuroGP*, pages 148–159. Springer-Verlag, 2007.
18. F. Jouault and J. Bézivin. KM3: A DSL for Metamodel Specification. In *FMOODS*, pages 171–185. Springer, 2006.
19. M. Kessentini, M. Wimmer, H. Sahraoui, and M. Boukadoum. Generating transformation rules from examples for behavioral models. In *BM-FA*, pages 2:1–2:7. ACM, 2010.
20. N. Lavrac and S. Dzeroski. *Inductive Logic Programming: Techniques and Applications*. Ellis Horwood, New York, 1994.
21. J.-N. Mazón and J. Trujillo. An mda approach for the development of data warehouses. *Decis. Support Syst.*, 45:41–58, 2008.
22. T. M. Mitchell. Generalization as search. *Artificial Intelligence*, 18:203–226, 1982.
23. S. Muggleton. Inductive Logic Programming. *New Generation Computing*, 8:295–318, 1991.
24. S. Muggleton. Optimal Layered Learning: A PAC Approach to Incremental Sampling. In *ALT*, pages 37–44, London, UK, 1993. Springer-Verlag.
25. S. Muggleton and L. D. Raedt. Inductive logic programming: Theory and methods. *J. Log. Program.*, 19/20:629–679, 1994.
26. S. Muggleton and K. Road. Predicate Invention and Utilisation. *Journal of Experimental and Theoretical Artificial Intelligence*, 6:6–1, 1994.
27. S. H. Nguyen, J. G. Bazan, A. Skowron, and H. S. Nguyen. Layered Learning for Concept Synthesis. *T. Rough Sets*, 3100:187–208, 2004.
28. R. Rios and S. Matwin. Predicate Invention from a Few Examples. In *AI*, pages 455–466, London, UK, 1998. Springer-Verlag.
29. A. Rutle, A. Rossini, Y. Lamo, and U. Wolter. A diagrammatic formalisation of mof-based modelling languages. In *TOOLS (47)*, pages 37–56. Springer, 2009.
30. A. Rutle, U. Wolter, and Y. Lamo. A diagrammatic approach to model transformations. In *EATIS*, 2008.
31. I. Stahl. On the Utility of Predicate Invention in Inductive Logic Programming. In *ECML*, pages 272–286. Springer, 1994.
32. I. Stahl. The Appropriateness of Predicate Invention as Bias Shift Operation in ILP. *Mach. Learn.*, 20:95–117, July 1995.
33. P. Stevens. Bidirectional model transformations in QVT: semantic issues and open questions. *Software and System Modeling*, 9(1):7–20, 2010.
34. P. Stone and M. M. Veloso. Layered learning. In *ECML*, pages 369–381. Springer, 2000.
35. M. Strommer and M. Wimmer. A framework for model transformation by-example: Concepts and tool support. In *TOOLS*, pages 372–391, Zurich, Switzerland, June 2008. Springer.
36. K. M. Ting and I. H. Witten. Stacked generalization: when does it work? In *IJCAI*, pages 866–871. Morgan Kaufmann, 1997.
37. K. M. Ting and I. H. Witten. Issues in stacked generalization. *J. Artif. Intell. Res. (JAIR)*, 10:271–289, 1999.
38. P. D. Turney. Exploiting context when learning to classify. In *ECML*, pages 402–407, London, UK, 1993. Springer-Verlag.
39. D. Varró. Model Transformation by Example. In *MoDELS*, pages 410–424, Genova, Italy, October 2006. Springer.
40. D. H. Wolpert. Stacked Generalization. *Neural Networks*, 5:241–259, 1992.
41. R. Wrembel and C. Koncilia. *Data Warehouses and OLAP: Concepts, Architectures and Solutions*. IGI Global, 2007.
42. Z. Xie. Several Speed-Up Variants of Cascade Generalization. In *FSKD*, pages 536–540, Xi'an, China, September 2006. Springer.
43. L. Zepeda, M. Celma, and R. Zatarain. A Mixed Approach for Data Warehouse Conceptual Design with MDA. In *ICCSA*, pages 1204–1217, Perugia, Italy, June 2008. Springer-Verlag.