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 interesting in the automation of *Model-Driven Data Warehouse* [6, 13, 2] using machine learning techniques. Intend is to automatically derive the transformations rules to be applied in the model-driven process. This aims to reduce the contribution of transformations designer and therefore 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 work in Predicate Invention, Layered Learning, Cascade Learning and Context Learning, we propose a new methodology that automatically updates the background knowledge of the concepts to be learned. Experimental results support the conclusion that this approach is suitable to solve this kind of problem.

1 Overview

The model-driven engineering [1] is a new promising approach for software development, which is mainly based on models, metamodels and transformations design. The models are conforming to metamodels and transformation rules are applied to refine them. The *Model-Driven Data Warehouse* represents approaches [6, 13, 2] that align the development of the data warehouse with a general model-driven paradigm. Transformation rules are the central components of the each model-driven process. However, transformations development is a very hard task because designers must have serious skills with the corresponding metamodels and the transformation languages. This make-up many risks and challenges during the transformations design and make the model-driven approach more complex and entails additional time and costs. One of the main challenges is to automatically learn these transformations from existing projects trace (or previous experiences). In this consideration, *Model Transformation By-Example* (introduced in [12]) is an active research area in model-driven software engineering that uses artificial intelligence techniques and proposes to automatically derive transformation rules.

Actually, in *Model-Driven Data Warehouse* context, several steps are needed to automatically learn the transformations rules. The first one consists in identifying the metamodels used to define the input/output models of these transformations rules and isolating steps where it is necessary to induce them. This is summarised by a previous works [2, 3] where a By-Example framework is proposed. We focus on transformations learning of a source-model (denoted DSPIM) to a target-model (denoted MDPIM). The *DSPIM* represents a conceptual view of the data source repository (the operational database) and the *MDPIM* represents a conceptual view of the data warehouse repository (the multidimensional database). We propose the *UML CORE* (part of the Unified Modelling Language metamodel[1]) and the *CWM OLAP* (part of the Common Warehouse Metamodel metamodel[2]), to design respectively, the source and the target models.

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* [8, 5] 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. In this paper, we investigate a new machine learning methodology stemming from the application needs: learning dependent-concepts. Following work about *Layered Learning* [10], *Predicate Invention* [9], *Context Learning* [11] and *Cascade Learning* [4], we propose a *Dependent-Concepts 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. Experimental evaluation show that the *DCL* system gives significantly better results.

The paper is organised as follows: Section 2 details the used machine learning algorithms and introduces the *Dependent-Concepts Learning* approach. Section 3 reports experimental results. Section 4 gives our concluding remarks.

2 Relational Learning of Dependent-Concepts

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 lan-

[1] http://www.omg.org/spec/UML
[2] http://www.omg.org/spec/CWM

guages 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 *Inductive Logic Programming* algorithms [8, 5] have been selected to deal with this learning problem. As ILP suffers from a scaling-up problem, the proposed architecture 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, presents the reduction of the *UML-CWM* problem to *ILP* and introduces the *DCL* approach.

2.1 Relational Learning Context

We consider the machine learning problem as defined in [7]. We assume that we are provided with i) a learning set $E = E^+ \cup E^-$ of positive (E^+) and negative examples (E^-) of a set of target concepts $\{c_1, \ldots, c_n\}$, drawn from an example language $\mathcal{L}_e$ ii) a hypothesis language $\mathcal{L}_h$, a generality relation $\geq$ that relates formulas of $\mathcal{L}_e$ and $\mathcal{L}_h$. For each target concept, the learning problem, defined as search in $\mathcal{L}_h$, is to find a hypothesis $h \in \mathcal{L}_h$ such that h is consistent with the data. A given hypothesis h is consistent if and only if it is both complete ($\forall e^+ \in E^+, h \geq e^+$) and correct ($\forall e^- \in E^-, h \not\geq e^-$). As explained in [7], 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[3] 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 $h \in \mathcal{L}_h$ 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 build a single clause of the target concept, Aleph selects an uncovered example as a seed, builds a most specific clause from this seed which is the lowest bound of the search space and then performs an admissible search over the space of clauses that subsume this lower bound according the user clause length bound. We have performed a detailed study on the reduction of the source model, target model and the mapping between them in ILP language. The main objective of this reduction is to use existing project traces to infer general transformation rules to be applied to future projects. As the used source and target models conform to their respective metamodels [3], the predicates selected are extracted from their respective metamodels (*UML CORE and CWM OLAP*). This work will be reported in the full-version of the paper.

[3] http://web.comlab.ox.ac.uk/oucl/research/areas/machlearn/Aleph

2.2 Dependent-Concepts Problem

Let $\{C_1, C_2, \ldots, C_n\}$ be the set of concepts to be learned in our problem. Each C_i for $i \in \{1, \ldots, n\}$ defines an ILP problem $(\vdash, \mathcal{B}_i, \mathcal{L}_{e_i}, \mathcal{L}_{h_i})$, where $\mathcal{B}_i \subseteq \mathcal{L}, \mathcal{L}_{e_i} \subseteq \mathcal{L}$ and $\mathcal{L}_{h_i} \subseteq \mathcal{L}$ are respectively language for background knowledge, examples language and hypotheses language for the concept C_i. An ILP learning algorithm is an algorithm which accept any $B \subseteq \mathcal{B}_i$, $E \subseteq \mathcal{L}_{e_i}$ as an input and computes $h \in \mathcal{L}_{h_i}$ if if h exists such that $B, h, E \nvdash \circ$, $B, h \vdash E^+$ and $(\forall e \in E^-)B, h \nvdash e$. If we consider all the concepts independently, each concept defines an independent *ILP* problem. First experiments in our system are done with this hypothesis. The second solution is to consider that concepts are dependent as explained below. We define a pre-order relation $\preceq$ between concepts such that $C_i \preceq C_j$ if the concept C_i depend on the concept C_j or if other things being equal, the quality of learning theory of C_i is better when the theory of C_j is known. As the dependence relation is given by the metamodels, the concepts must be organised according to the given preorder relation. A concept C_i is called child of the concept C_j (or C_j parent of C_i) if and only if $C_i \preceq C_j$ and there exist no concept C_k such that $C_i \preceq C_k \preceq C_j$. A concept C_j is called root concept iff there exists no concept C_k such that $C_j \preceq C_k$. An ILP learning algorithm of set of dependent concepts is an algorithm which accepts a preorder set of concepts, starts with learning root concepts and propagates the learned concepts to the background knowledge of their child and continue recursively the learning process until the resolution of the learning problem.

3 Empirical Results

This section describes the experimental setup and compares the results of the two tested methods:

1. The *Independent Concept Learning (ICL)* approach which proposes to learn the set of considered concepts independently.
2. The *Dependent Concept Learning (DCL)* approach as it is described in the previous section explores a dependency graph to learn the concepts. Within this approach, we benchmark two settings: (i) the background knowledge B of dependent concepts is updated with parent concept instances (denoted *DCLI*) and (ii) with parent concept intensional definitions (denoted *DCLR*). We identify the following concepts dependencies: $ClassToCube \rightarrow PropertyToMeasure$; $ClassToCube \rightarrow RelationShipToDimension$; and $ClassToCube \rightarrow RelationShipToDimension \rightarrow ClassToLevel$.

The first goal of this analysis is to examine if the choice of the number of training models and examples influence the performances. We report the obtained test accuracy ($Accuracy = TP + TN/P + N$) curves (figure 1) of learning *ClassToCube* and *PropertyToDimension* concepts (concepts of level 1 in the dependency graph). The second goal of the analysis is to study the performances of the *DCL* approach (with the two settings) compared to the *ICL* approach. We

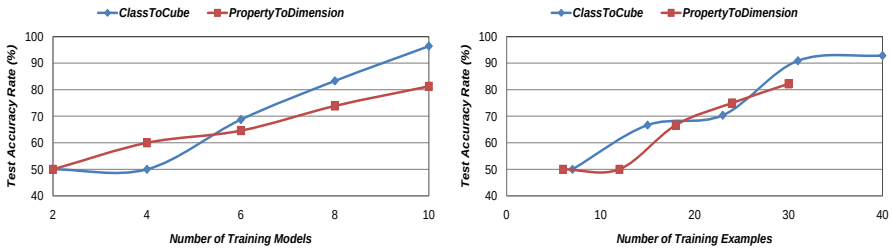

Fig. 1. Accuracy for *ClassToCube* and *PropertyToDimension*.

report the *Receiver-Operating-Characteristics (ROC)* curves (figures 2 and 3) of the tested approaches (*ICL, DCLI and DCLR)* for learning *PropertyToMeasure*, *RelationshipToDimension* (concepts of level 2) and *ClassToLevel* concepts (concept of level 3). For all experiments, we use the repeated random sub-sampling validation strategy (the tests average of 10 iterations is reported). Due to space constraint, we will omit results discussion; however they will be available in the full-version of the paper. ROC graphs are two-dimensional graphs in which *tp rate (sensitivity)* is plotted on the Y axis and *fp rate (1 - specificity)* is plotted on the X axis. The *true-positive-rate* (also called hit rate and recall) and the *false-positive-rate* (also called false alarm rate) are estimated as: $tp\ rate = TP/P$ and $fp\ rate = FP/N$.

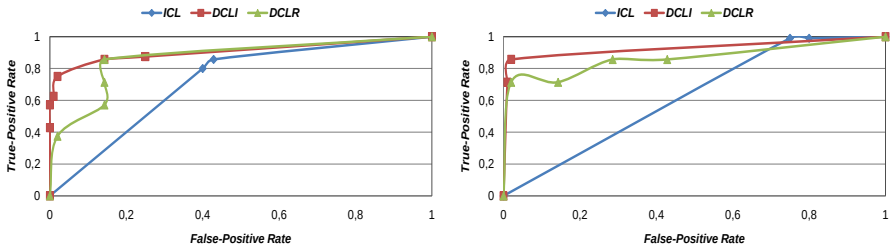

Fig. 2. Learning *PropertyToMeasure* and *RelationshipToDimension* ROC Curves.

The area under the ROC curve, abbreviated AUC, is the common measure to compare the tested methods. The AUC represents a measure of accuracy. Results in figures 2 and 3 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; it is the less accurate approach (random performance curve). In the case of *DCLR* setting, results are less than the *DCLI* setting, but it present good performances compared to *ICL* approach. Then, we examine the gain in accuracy of the *DCLR and ICL*. Let $GA^C_{DCLR-ICL} = AUC^C_{DCLR} - AUC^C_{ICL}$ be the gain in accuracy (of *DCLR* compared to *ICL*) when learning the concept C. For *PropertyToMeasure* and *RelationshipToDimension*: $GA^{PropertyToMeasure}_{DCLR-ICL} < GA^{RelationshipToDimension}_{DCLR-ICL}$. This will be discussed regarding the dependency on *ClassToCube*.

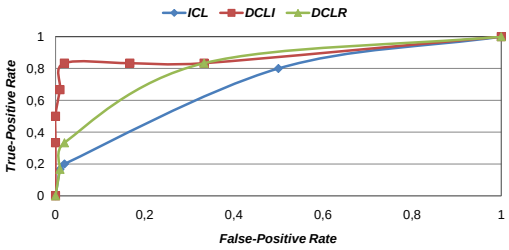

Fig. 3. ROC Curves of Learning $ClassToLevel$.

4 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 an 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-concepts. Experimental results show that the proposed *DCL* 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.

References

1. J. Bézivin. Model driven engineering: An emerging technical space. In *GTTSE*, pages 36–64. Springer, 2006.
2. 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.
3. M. Essaidi and A. Osmani. Towards Model-driven Data Warehouse Automation using Machine Learning. In *IJCCI (ICEC)*, pages 380–383, Valencia, Spain, 2010. SciTePress.
4. J. Gama. Combining Classifiers by Constructive Induction. In *ECML*, pages 178–189. Springer, 1998.
5. N. Lavrac and S. Dzeroski. *Inductive Logic Programming: Techniques and Applications*. Ellis Horwood, New York, 1994.
6. J.-N. Mazón and J. Trujillo. An mda approach for the development of data warehouses. *Decis. Support Syst.*, 45:41–58, 2008.
7. T. M. Mitchell. Generalization as search. *Artificial Intelligence*, 18:203–226, 1982.
8. S. Muggleton and L. D. Raedt. Inductive logic programming: Theory and methods. *J. Log. Program.*, 19/20:629–679, 1994.
9. S. Muggleton and K. Road. Predicate Invention and Utilisation. *Journal of Experimental and Theoretical Artificial Intelligence*, 6:6–1, 1994.
10. P. Stone and M. M. Veloso. Layered learning. In *ECML*, pages 369–381. Springer, 2000.
11. P. D. Turney. Exploiting context when learning to classify. In *ECML*, pages 402–407, London, UK, 1993. Springer-Verlag.
12. D. Varró. Model Transformation by Example. In *MoDELS*, pages 410–424, Genova, Italy, October 2006. Springer.
13. 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.