



TRANSFER LEARNING BY MAPPING AND REVISING BOOSTED RELATIONAL DEPENDENCY NETWORKS

Rodrigo Azevedo Santos

Dissertação de Mestrado apresentada ao Programa de Pós-graduação em Engenharia de Sistemas e Computação, COPPE, da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Mestre em Engenharia de Sistemas e Computação.

Orientadores: Gerson Zaverucha
Aline Marins Paes Carvalho

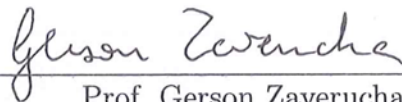
Rio de Janeiro
Junho de 2019

TRANSFER LEARNING BY MAPPING AND REVISING BOOSTED
RELATIONAL DEPENDENCY NETWORKS

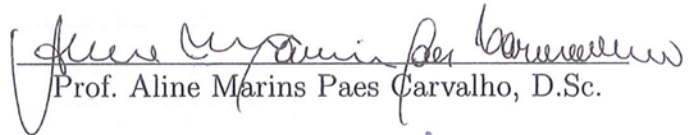
Rodrigo Azevedo Santos

DISSERTAÇÃO SUBMETIDA AO CORPO DOCENTE DO INSTITUTO
ALBERTO LUIZ COIMBRA DE PÓS-GRADUAÇÃO E PESQUISA DE
ENGENHARIA (COPPE) DA UNIVERSIDADE FEDERAL DO RIO DE
JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A
OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA DE
SISTEMAS E COMPUTAÇÃO.

Examinada por:



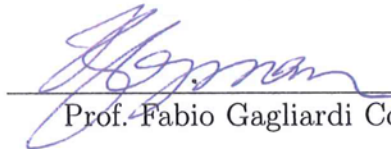
Prof. Gerson Zaverucha, Ph.D.



Prof. Aline Marins Paes Carvalho, D.Sc.



Prof. Valmir Carneiro Barbosa, Ph.D.



Prof. Fabio Gagliardi Cozman, Ph.D.

RIO DE JANEIRO, RJ – BRASIL

JUNHO DE 2019

Azevedo Santos, Rodrigo

Transfer Learning by Mapping and Revising Boosted
Relational Dependency Networks/Rodrigo Azevedo
Santos. – Rio de Janeiro: UFRJ/COPPE, 2019.

XIII, 74 p.: il.; 29, 7cm.

Orientadores: Gerson Zaverucha

Aline Marins Paes Carvalho

Dissertação (mestrado) – UFRJ/COPPE/Programa de
Engenharia de Sistemas e Computação, 2019.

Bibliography: p. 66 – 74.

1. Transfer learning. 2. Statistical relational
learning. 3. Theory revision. I. Zaverucha, Gerson
et al. II. Universidade Federal do Rio de Janeiro, COPPE,
Programa de Engenharia de Sistemas e Computação. III.
Título.

*Dedico este trabalho a minha
mãe Lidia e a minha avó Ilma
pelo imenso orgulho e carinho
que sentiram por mim.*

Agradecimentos

Agradeço a minha amada mãe Lidia Azevedo (in memoriam) que me ensinou o gosto pela leitura e aprendizado, e pelo imenso orgulho que sentiu ao me ver ingressar no curso de mestrado. A você, que sempre compreendeu momentos de minha ausência devido aos estudos, partilho este momento de alegria com muito amor e saudade.

Agradeço a minha avó Ilma Gonçalves (in memoriam) pelo apoio aos meus estudos durante toda sua vida e todo o carinho e orgulho que sempre sentiu por mim. O meu eterno amor e gratidão.

Ao meu pai Alexis de Souza pela educação que pôde me proporcionar e a todos os meus familiares e amigos por todo apoio e dedicação.

Agradeço aos meus orientadores Gerson Zaverucha e Aline Paes que aceitaram me orientar e foram essenciais para a realização deste trabalho. Agradeço a disponibilidade para as reuniões e discussões acerca do desenvolvimento do trabalho e a todo o conhecimento e suporte dado a mim durante este período.

Aos professores Valmir Barbosa e Fábio Cozman pela participação da banca e pela contribuição dada através dos comentários sobre o trabalho.

Aos professores do Programa de Engenharia de Sistemas e Computação, pois foram essenciais para o aprendizado e minha formação. Assim como a equipe administrativa e de suporte técnico que sempre foram muito solícitos.

Aos meus colegas de curso, pelo bom convívio e troca de conhecimento durante todo esse tempo.

À comunidade científica e principalmente aos autores dos trabalhos relacionados a esta dissertação que também contruíram com a disponibilização de seus algoritmos.

À Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) pelo suporte financeiro que viabilizou este trabalho.

Resumo da Dissertação apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M.Sc.)

TRANSFERÊNCIA DE APRENDIZADO AO MAPEAR E REVISAR REDES DE DEPENDÊNCIA RELACIONAL COM BOOSTING

Rodrigo Azevedo Santos

Junho/2019

Orientadores: Gerson Zaverucha

Aline Marins Paes Carvalho

Programa: Engenharia de Sistemas e Computação

Algoritmos de aprendizado de máquina normalmente assumem que há disponível uma quantidade considerável de dados para a realização do treinamento de modelos. Com isso, as abordagens tradicionais falham em abordar domínios onde dados são difíceis ou custosos de se obter. A transferência de aprendizado surgiu para abordar o problema de escassez de dados ao considerar um modelo aprendido em um domínio de origem, onde dados são fáceis de se obter, como um ponto inicial para o domínio alvo. Por outro lado, dados de um mundo real são compostos por objetos e suas relações que normalmente advêm de ambientes com ruído. Encontrar padrões em dados relacionais probabilísticos tem sido o foco da área de Aprendizagem Estatística Relacional. Para abordar as questões de dados escassos, relacionais e incertos, neste trabalho propusemos o TreeBoostler, um algoritmo que transfere Redes de Dependência Relacional com boosting aprendidas em um domínio de origem para um domínio alvo. O algoritmo TreeBoostler primeiramente encontra um mapeamento entre pares de predicados para representar as árvores no vocabulário alvo. Após, o algoritmo aplica dois operadores de revisão de teoria para modificar a árvore de regressão relacional com o intuito de lidar com incorreções e melhorar o desempenho das árvores mapeadas. Os resultados mostraram que o TreeBoostler foi capaz de transferir conhecimento entre diversos domínios distintos com sucesso, além de ter mostrado um desempenho comparável ou melhor que os métodos de aprendizado do zero em termos de acurácia e ter obtido um desempenho melhor em termos de acurácia e tempo de execução comparado a um método de transferência de aprendizado disponível na literatura.

Abstract of Dissertation presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Master of Science (M.Sc.)

TRANSFER LEARNING BY MAPPING AND REVISING BOOSTED RELATIONAL DEPENDENCY NETWORKS

Rodrigo Azevedo Santos

June/2019

Advisors: Gerson Zaverucha

Aline Marins Paes Carvalho

Department: Systems Engineering and Computer Science

Statistical machine learning algorithms usually assume that there is considerably-size data to train the models. However, traditional approaches fail to address domains where data is difficult or expensive to obtain. Transfer learning has emerged to address this problem of data scarcity by relying on a model learned in a source domain where data is easy to obtain to be a starting point for the target domain. On the other hand, real-world data is composed of objects and their relations usually disposed of in a noisy environment. Finding patterns through such uncertain relational data has been the focus of the Statistical Relational Learning area. To address these issues, scarce, relational, and uncertain data, in this work we propose TreeBoostler, an algorithm that transfers Boosted Relational Dependency Networks learned in a source domain to the target domain. TreeBoostler first finds a mapping between pairs of predicates to accommodate the trees in the target vocabulary. Then, it employs two novel theory revision operators devised to change relational regression trees to handle incorrectness and improve the performance of the mapped trees. TreeBoostler has successfully transferred knowledge among several distinct domains. It performs comparably or better than learning from scratch methods in terms of accuracy and outperforms an existing transfer learning approach in terms of accuracy and runtime.

Contents

List of Figures	x
List of Tables	xii
1 Introduction	1
1.1 Contributions	2
1.2 Outline	3
2 Background	5
2.1 Relational Learning	5
2.1.1 First-Order Logic	6
2.1.2 Inductive Logic Programming	11
2.1.3 Statistical Relational Learning	18
2.2 Theory Revision	20
2.2.1 Definition	21
2.2.2 Revision Points	22
2.2.3 Revision Operators	23
2.2.4 FORTE Algorithm	24
2.3 Transfer Learning	25
2.3.1 Definition	27
2.3.2 Research issues	27
2.3.3 Scenarios	28
2.3.4 Taxonomy	29
2.3.5 Approaches	30
2.4 RDN-Boost	32
2.4.1 Relational Dependency Networks	32
2.4.2 Functional Gradient Boosting	34
2.4.3 Relational Regression Trees	36
2.4.4 Learning algorithm	38
2.5 Related work	39
2.6 Final remarks	40

3	TreeBoostler: The proposed algorithm	41
3.1	Transferring the structure	41
3.1.1	Legal mappings	42
3.1.2	Finding best mapping and transferring the structure	43
3.2	Revising the structure	45
3.2.1	Pruning	48
3.2.2	Expansion	49
3.3	Final remarks	50
4	Experiments and results	52
4.1	Research questions	52
4.2	Datasets	53
4.3	Methodology and results	54
4.4	Final remarks	63
5	Conclusion	64
5.1	Future work	65
	Bibliography	66

List of Figures

2.1	SLD-Resolution tree for the query path(a,d)	12
2.2	Demonstration of completeness and consistency of a hypothesis	14
2.3	Taxonomy of theory refinement tasks	22
2.4	Traditional machine learning setup	26
2.5	Transfer learning setup	26
2.6	Taxonomy of transfer learning settings	31
2.7	(a) An example of Bayesian network. (b) The corresponding dependency network example	33
2.8	Example of RDN for a university domain	34
2.9	Example of RPT	34
2.10	Example of RRT	37
3.1	One regression tree to be transferred from UW-CSE to Cora for query predicate <i>advisedby</i> . Regression values are not considered for transfer. They are relearned in the process.	45
3.2	The transfer learning process stages. The trees presented are the following: obtained from source domain by learning from scratch (top-left); transferred by mapping predicates (top-right); after the pruning process (down-left) and after the expansion of nodes (down-right). All trees are the first one learned in the iterations. The transference is done from IMDB to UW-CSE and depth limits were reduced to generate smaller trees. Regression values are not considered in the pruning process and they are relearned when expanding nodes.	51
4.1	Learning curves for AUC ROC (left) and AUC PR (right) obtained from IMDB \rightarrow UW-CSE.	58
4.2	Learning curves for AUC ROC (left) and AUC PR (right) obtained from NELL Sports \rightarrow NELL Finances.	58
4.3	Learning curves for AUC ROC (left) and AUC PR (right) obtained from Yeast \rightarrow Twitter.	59

4.4	Learning curves for AUC ROC (left) and AUC PR (right) obtained from Twitter → Yeast.	59
4.5	Learning curves for AUC ROC (left) and AUC PR (right) obtained from IMDB → Cora.	59
4.6	Learning curves for AUC ROC (left) and AUC PR (right) obtained from Cora → IMDB.	60
4.7	Learning curves from minimal target data for AUC ROC (left) and AUC PR (right) obtained from IMDB → UW-CSE.	61
4.8	Learning curves from minimal target data for AUC ROC (left) and AUC PR (right) obtained from NELL Sports → NELL Finances.	61
4.9	Learning curves from minimal target data for AUC ROC (left) and AUC PR (right) obtained from Yeast → Twitter.	62
4.10	Learning curves from minimal target data for AUC ROC (left) and AUC PR (right) obtained from Twitter → Yeast.	62
4.11	Learning curves from minimal target data for AUC ROC (left) and AUC PR (right) obtained from IMDB → Cora.	62
4.12	Learning curves from minimal target data for AUC ROC (left) and AUC PR (right) obtained from Cora → IMDB.	63

List of Tables

2.1	Example of an ILP problem. The idea is to learn the predicate <i>daughter</i> . Positive examples are denoted by \oplus and negative by \ominus . . .	17
2.2	An example of MLN with two formulas and their respective weights . . .	20
2.3	Different settings of transfer learning	30
3.1	Found predicate mapping for transferring IMDB→UW-CSE	43
4.1	Statistics about datasets	54
4.2	Results on IMDB and Cora dataset. We compare our algorithm, RDN-B (that uses boosting), RDN and TODTLER. We present the results for the area under curves for ROC and PR and the conditional log-likelihood for test examples. We also present the training time. . .	57
4.3	Results on Yeast and Twitter dataset. We present the results for the area under curves for ROC and PR, the conditional log-likelihood and the training time.	57
4.4	Results on transference from IMDB to UW-CSE dataset and NELL Sports domain to Finances domain considering area under the curves for ROC and PR, the conditional log-likelihood and the training time. . .	57

List of Algorithms

1	FORTE Algorithm	24
2	RDN-Boost algorithm	38
3	Finding legal mappings given source and target predicates	43
4	Top-Level Transfer Algorithm	46
5	Top-Level Theory Revision Algorithm	49
6	Pruning Operator: Removes nodes recursively if they fit the definition of Revision Point	49
7	Expansion Operator: Performs expansion of nodes	50

Chapter 1

Introduction

Machine learning algorithms have been widely and successfully used in many areas such as computer vision, robotics, social network analysis, and others [1, 2]. However, this success usually comes with the presence of large amounts of data. When the number of examples is relatively small, learning good models can be a challenging task. This is often the case of several real-world problems where collecting data is expensive or even impossible to obtain, such as collecting data from movements of real-world robots [3], collecting WiFi signal data from a large number of locations [4] and labeling data for sentiment classification [5]. To handle this issue, transfer learning techniques [6] leverage a model learned from a source domain with more examples to learn from another, related, domain where data is more scarce.

Transfer learning has been widely employed in classical machine learning settings, such as ensembles [7] and decision trees [8]. However, most of them do not take into account the relationships between entities of the domain and the fact that the examples may not be identically and independently distributed, which is the case of a number of real-world data such as interaction between proteins [9] and interaction between accounts in social media [10]. In addition, real-world data have noise and are generally uncertain. This is the focus of the area called Statistical Relational Learning (SRL) [11]. Transfer Learning algorithms have also been developed in the context of SRL. Two of these algorithms [10, 12] transfer relational knowledge by creating a second-order representation of formulas from learned Markov Logic Networks (MLN) [13]. Other three algorithms [14-16] find predicate mappings through search methods to perform transference of clauses learned from MLNs by mapping their predicates.

Although these methods showed better results compared to MLN models learned from scratch, NATARAJAN *et al.* [17] have shown that applying a boosted approach to learn Relational Dependency Networks (RDN) yielded superior performance over traditional SRL approaches. Based on the predicate mapping algorithm presented by MIHALKOVA *et al.* [14] to transfer MLN clauses, we developed a similar predi-

cate mapping approach to perform transference of Boosted RDNs, which are models that have a higher expressiveness. We have opted for using Boosted RDNs as the models to be transferred due to its efficiency in both training and inference time and the capability of learning both the structure and the parameters of RDNs simultaneously, which is not the case of the MLN models used by related work. RDN-Boost has been shown to have state-of-the-art results in learning RDNs and superior performance over other SRL models in much less training time.

In this dissertation, motivated by the need of learning from scarce, relational and uncertain data, we present a transfer learning algorithm called TreeBoostler that transfers Boosted RDNs by mapping the predicates appearing in the trees. At a higher level, the algorithm generates the possible predicate mappings as it tries to recursively transfer nodes from the source regression trees. After finding such mappings, they are propagated to the rest of the tree and the other trees of the next iterations. To complement the process and better adjust the mapped trees to the new, target domain, TreeBoostler also includes a theory revision [18] algorithm for proposing modifications to the mapped models in order to handle incorrectness and to improve the performance.

We evaluated TreeBoostler in several real-world datasets and simulated the scenario where only a few data are available by training on one single fold and testing on the remaining folds. Our results demonstrate that our method has successfully transferred learned knowledge across different domains in a smaller time compared to other transfer learning algorithms. In addition, transference showed to be very useful in terms of accuracy compared to learning from scratch methods based on RDNs. Additional experiments were performed to investigate the behavior of the algorithm as the number of examples increases and when provided minimal target data. The results demonstrate that our algorithm can be very competitive to traditional methods that learn from scratch even with the increase of the amount of data, also when provided only a few examples.

1.1 Contributions

To sum up, the main contributions of this dissertation include:

- A method, namely TreeBoostler, that transfers learned boosted RDNs between related domains.

We proposed the TreeBoostler, a transfer learning algorithm that constructs a target set of relational regression trees biased by a predicate mapping found through the transfer process given the structure of the source regression trees. This predicate mapping is found by applying all legal mappings to a node and

selecting the one which gives the best split.

- A revision theory system that proposes modifications to boosted trees.

We proposed two revision operators to the revision theory system implemented in TreeBoostler. These revision operators are: (1) pruning operator, which increases the coverage of examples by deleting nodes from a tree and (2) expansion operator, which decreases the coverage of examples by expanding nodes in each tree.

- Extensive experiments to evaluate our proposed method.

We conducted three types of experiments to evaluate TreeBoostler against baseline approaches. The experiments were conducted in the following way: (1) simulating a transfer learning environment with limited target data, (2) providing to the system a scenario with increasing amounts of target data and (3) providing a scenario with learning from minimal target data, i.e. very few examples.

1.2 Outline

The remainder of this dissertation is organized as follows:

In Chapter 2, we introduce the background information that is relevant in order to understand the contents of this work. We review the fundamentals of relational learning such as First-Order Logic, Inductive Logic Programming and Statistical Relational Learning. Then, we define theory revision and transfer learning and present a taxonomy for both concepts. We furthermore review the algorithm for boosting RDNs and the concepts necessary for its understanding which are Relational Dependency Networks, Functional Gradient Boosting and Relational Regression Trees. We then discuss related work and point out the similarities among the algorithms.

In Chapter 3, we present the algorithm TreeBoostler and its process of transfer and revision. We propose two approaches to perform a complete transfer system. The first approach performs mapping of predicates by founding the best legal mapping for each node while the second approach proposes two modifications to the mapped models which are pruning and expansion of nodes.

In Chapter 4, we provide the results of the proposed transfer learning algorithm applied in several different datasets. We compare our results against related works for transfer learning and traditional learning from scratch methods in an experiment that simulates a transfer learning environment with limited target data. We also compare the results of our algorithm against traditional learning from scratch methods in two experiments considering a scenario with increasing amounts of target data

and a scenario that represents learning from minimal target data.

Finally, in Chapter 5, we conclude with remarks and present possible directions for future research.

Chapter 2

Background

In this chapter, first, we briefly introduce the reader about the problem of learning from relational data and present the fundamentals of the research area. After that, we outline the use of functional-gradient boosting to learn relational dependency networks, which is the base algorithm for our proposed work. Third, we describe the concepts related to theory revision. Finally, we give a comprehensive overview of transfer learning and related works.

2.1 Relational Learning

The majority of machine learning algorithms assume data as points in a high-dimensional space. For example, pictures are represented by pixels in the task of classifying or detecting objects in images; words can be represented by its number of occurrences in a text in the task of classifying web pages. The examples, in this assumption, have a vector representation in which each feature has a corresponding value.

These machine learning algorithms such as neural networks, decision trees and linear models focus on the attribute-value representation and ignore relational aspects of data losing useful knowledge [11]. Such relational knowledge may help, considering, for example, the recommendation task, to detect not only if a movie should be recommended given its characteristics (e.g. duration, genre, etc) but also given the information of how related people (i.e. friends) liked a particular movie or how related people liked related movies. Thus, in relational data, we are also interested in taking advantage of its logical structure, which contains crucial information that determines how objects participate in relationships and events, in order to solve more complex problems.

Roughly speaking, we may say that traditional learning is done given one single table while relational learning algorithms may rely on multiple tables in a database. In order to apply traditional learning algorithms to a data set consisting of multiple

tables, a preprocessing step is needed to integrate the data into a single table. This process may produce very sparse datasets or cause loss of information due to important relations that involve objects.

To deal with the object-relational structure of the data many algorithms have been developed to learn first-order rules from relational data and reach conclusions about related objects [19, 20]. Also, several approaches attempt to model complex and probabilistic structure of relational domains, which basically differs from the deterministic approaches by dealing with uncertainty in data. We will give the details to understand these approaches in the following subsections.

2.1.1 First-Order Logic

A powerful way to describe and represent relationships among objects is with the use of First-order logic (FOL). Relational data sets can be represented by logical facts containing predicates and terms that describe how objects interact with each other. With FOL, we allow variables to be bound to atomic symbols and because of that we can construct very useful rules that describe a pattern in a domain, also we can obtain answers regarding the data through queries.

The basic concepts include language (syntax), model (semantics) and proof theory (deductive system). The syntax of logic programs is concerned with what are the legal statements in the language. On the other hand, the semantics is concerned with assigning meaning to such statements [11].

The syntax of FOL consists of the following symbols:

- Quantifiers
- Connectives
- Variables
- Constants
- Predicates
- Functions

Predicates represent relations between objects in the domain, such as a relation *publishedby* that may connect the entities paper and person (i.e. a paper is published by a person). Commonly, when representing relational data using FOL, arguments may be associated with a type, for efficiency, which are paper and person in the previous example. A predicate is represented by a *name* and the number of arguments a predicate takes is its arity. We usually refer to a predicate by *name/n*

where n is its arity. We call binary predicates those predicates that have an arity 2; similarly, we call unary predicates those predicates that describe properties (arity 1). Predicates can also represent properties of objects which distinguish them from each other (e.g. student, actor, blue, large, etc).

These objects (or entities) are represented by constants. They are individuals in a specific world (e.g. persons, cars, papers, movies, teams, companies, etc). Along with the predicates, we can say that John has a dog through the atom $haspet(john, dog)$ and that John is a tall person (property) through $tall(john)$. Variables can be assigned to quantified formulas in order to be substituted by constants or function terms. The substitution of variables is very useful to answer queries about the domain.

A function maps a set of inputs consisted of individuals to an output consisted of one single individual. Function symbols are followed by a bracket n -tuple of terms (e.g. $f(X_1, X_2)$). A term can be a variable, a constant or a function symbol applied to a set of terms; for instance, X , $f(X_1, \dots, X_n)$ and $john$ are terms. A term with no variables is a ground term. We adopt the Prolog [21] syntax and start variables names with capital letters; constants are in lowercase.

An atom is a predicate applied to terms. It describes the relation between objects or describes a property of an object represented by a term. An atom can be composed of a predicate symbol followed by a bracketed n -tuple of terms. For example, $mother(mary, john)$, $father(X, Y)$ and $married(john, X)$ are atoms. An atom whose all terms are constants is called a ground atom (e.g. $mother(mary, john)$).

A literal can be either an atom or a negated atom. Thus, considering the semantics, a literal can have true value if it can be proved; in the case of the negation of an atom, a literal can have true value if it has failed to be proved.

The remaining syntax symbols, quantifiers and connectives, are used to form more complex formulas. Formulas, also called sentence or statement, are either atoms or one of the following forms: F , (F) , $\neg F$, $F \vee G$, $F \wedge G$, $F \leftarrow G$, $F \leftrightarrow G$, $\forall X : F$ and $\exists X : F$, where F and G are formulas and X is a variable. The connectives are the following:

- Negation
- Disjunction
- Conjunction
- Implication
- Equivalence

Considering the formulas F and G , we next describe each connective.

The negation (\neg) describes the negation of a formula. For instance, the formula $\neg F$ denotes the negation of F . Thus, $\neg F$ is true whenever F is false. The disjunction (or), denoted by \vee , states that a formula is true, for instance $F \vee G$, if and only if one or more of its operands is true. On the contrary, the conjunction (and), denoted by \wedge , states that a formula is true if and only if all of its operands are true. For example, $F \wedge G$ is true only if both F and G are true.

The implication, denoted by an arrow, is used like the following formulas $F \leftarrow G$ or $G \rightarrow F$. It describes that F if G , which is equivalent to the formula $F \vee \neg G$. Thus, the truth-value is false if and only if G is true and F is false. The statement $G \rightarrow F$ can be read as “ G implies F ” and means that if G is true, then F must also be true. The statement can also be read as “if G then F ”.

Equivalence is denoted by a bidirectional arrow and states that $F \leftrightarrow G$ is true whenever F and G have the same truth-value. In short, F if and only if G . The equivalence connective is the formula $(F \rightarrow G) \wedge (G \rightarrow F)$.

Finally, we define the two following quantifiers:

- Universal quantifier
- Existential quantifier

These quantifiers are useful to quantify variables and therefore objects in a world. They allow reasoning about multiple objects simultaneously.

The universal quantifier corresponds to the phrase “for all” and is denoted by \forall . Thus, for a given formula F and a variable X , we say that for all X the formula F holds. Example [2.1.1](#) shows two statements with this quantifier.

Example 2.1.1 *Consider the following formulas. The first one states that every dog is an animal. The second statement says that every even number is not an odd number.*

$$\forall X \text{ dog}(X) \rightarrow \text{animal}(X)$$

$$\forall X \text{ even}(X) \rightarrow \neg \text{odd}(X)$$

The existential quantifier corresponds to the phrase “there exists” and is denoted by \exists . Thus, for a given formula F and a variable X , we say that there exists a X which F is true. Example [2.1.2](#) shows two statements with this quantifier.

Example 2.1.2 *Consider the following formulas. The first statement says that exists at least one animal that is mammal and oviparous (animals that lay eggs). The second statement says that some numbers are even and prime.*

$$\exists X \text{ mammal}(X) \wedge \text{oviparous}(X)$$

$$\exists X \text{ even}(X) \wedge \text{prime}(X)$$

A clause is a disjunction of literals preceded by a universal quantifier for one of each variable presented in the disjunction of literals. To sum up, a clause is a formula of the following form $\forall X_1 \forall X_2 \dots \forall X_m (L_1 \vee L_2 \vee \dots \vee L_n)$, where each L_i is a literal and each X_i is a variable occurring in the literals $(L_1 \vee L_2 \vee \dots \vee L_n)$.

A clause can also be represented as a set of literals in the form $\{L_1, L_2, \dots, L_n\}$ which stands for the clause $L_1 \vee L_2 \vee \dots \vee L_n$. We can also represent the clause in a form of an implication formula. For example, $\{A_1, A_2, \dots, A_n, \neg B_1, \neg B_2, \dots, \neg B_m\}$ can be represented with the equivalent clause $A_1 \vee A_2 \vee \dots \vee A_n \leftarrow B_1 \wedge B_2 \wedge \dots \wedge B_m$. The same clause is commonly written as $A_1, A_2, \dots, A_n \leftarrow B_1, B_2, \dots, B_m$ by omitting disjunctions and conjunctions. The literals B_1, B_2, \dots, B_m are called the body and A_1, A_2, \dots, A_n the head of the clause.

A Horn clause is a clause that contains at most one positive literal and a definite clause is a clause with exactly one positive literal. A definite clause is an expression of the form $H \leftarrow B_1, \dots, B_n$ where H is a literal (head) and B_1, \dots, B_n are literals that forms the body of the clause. All variables are implicitly universally quantified. A fact is a clause whose body is empty, thus a fact consists of a single positive literal (e.g. $\text{parent}(\text{tom}, \text{peter}) \leftarrow$). Arrows are generally omitted in facts.

Finally, a logic program is defined as a finite set of definite clauses. For more details, please refer to [22]. We illustrate these concepts on an example.

Example 2.1.3 Consider the following example of logic program that represents people of a particular family.

```

female(ilm)
female(lid)
parent(lid, rod)
parent(ilm, lid)
grandmother(A, B) ← parent(A, C), parent(C, B), female(A)

```

In Example 2.1.3, the following is a definite clause:

$$\text{grandmother}(A, B) \leftarrow \text{parent}(A, C), \text{parent}(C, B), \text{female}(A)$$

This definite clause represents the rule that expresses when a person A is grandmother of B .

In Example 2.1.3, the following is a fact:

$$\text{parent}(\text{lid}, \text{rod})$$

This fact states that the person *lid* is parent of *rod*. Given these facts and the rule, we can conclude that *ilm* is grandmother of *rod*.

A clause is called function-free if it contains no function symbols and constants (e.g. the rule in Example 2.1.3). Commonly, in relational learning for simplicity, only variables and constants are considered and can be used as predicate arguments.

A substitution $\theta = \{V_1/t_1, \dots, V_n/t_n\}$ is applied to terms, atoms or clauses in order to replace simultaneously all occurrences of the variables V_i to their respective terms t_i . Example 2.1.4 demonstrates applying a substitution to a clause.

Example 2.1.4 Consider the last clause presented in Example 2.1.3. The substitution $\theta = \{A/ilm, B/rod, C/lid\}$ applied to this clause results in the following:

$$grandmother(ilm, rod) \leftarrow parent(ilm, lid), parent(lid, rod), female(ilm)$$

Two clauses or terms c_1 and c_2 can be *unified* if there exists a substitution θ_1 applied to c_1 and a substitution θ_2 applied to c_2 such that $c_1\theta_1 = c_2\theta_2$. A substitution θ is called the *most general unifier*, denoted by $mgu(c_1, c_2)$, of atoms (or clauses) c_1 and c_2 if and only if $c_1\theta = c_2\theta$. In other words, the most general unifier is the simplest substitution applied to make both atoms equal.

Applying these substitutions is very useful for reasoning which can be performed by SLD-Resolution (Selective Linear Definite clause resolution). Considering the knowledge as a logic program (composed of definite clauses), SLD-Resolution is a feasible way of inferring whether a query holds. The task of inference is to determine whether a query is true in the least Herbrand model of a logic program.

The Herbrand base of a logic program is the set of all ground atoms formed through predicate symbols, functions and constants occurring in the program. Herbrand interpretations are subsets of the Herbrand base. A Herbrand interpretation is a model of a clause $h \leftarrow b_1, \dots, b_n$ if for every substitution θ such that all $b_i\theta$ are in the interpretation, $h\theta$ is also in the interpretation. The least Herbrand model is the set of all ground atomic logical consequences of the program. Example 2.1.5 demonstrates a Herbrand base of a logic program.

Example 2.1.5 Consider a simple logic program that contains the clauses $f(0)$ and $f(s(X)) \leftarrow f(X)$. The following Herbrand base contains all atoms that can be built from the predicate symbol $f/1$, function symbol $s/1$ and constant 0

$$hb = \{ f(0), f(s(0)), f(s(s(0))), \dots \}$$

In our example, the query $\leftarrow f(0)$ has a true answer, while $\leftarrow f(1)$ has a false answer. In this case, we say the query fails. We can also query the data using variables in the arguments. Suppose we would like to know *rod*'s parent in Example 2.1.3. This query could be represented by the following:

$\leftarrow \text{parent}(X, \text{rod})$

If a query is not ground, inference asks for the existence of a substitution that grounds the query into an atom that is part of the least Herbrand model, i.e. inference asks which constant may replace the variable X in order to find a clause in a logic program. In this example, $\theta = \{X/\text{lid}\}$ is a substitution for the query.

To answer such queries, resolution is performed in order to derive the empty clause. More specifically, resolution starts a SLD-tree with a root that corresponds to the query. The process continues until it reaches the empty clause. Each branch is considered a derivation and derivations that result in the empty clause are called proofs.

Example 2.1.6 *The following program represents a graph with four nodes and defines paths between nodes in terms of edges.*

$c_1 : \text{path}(X, Y) \leftarrow \text{edge}(X, Y)$
 $c_2 : \text{path}(X, Y) \leftarrow \text{edge}(X, Z), \text{path}(Z, Y)$
 $c_3 : \text{edge}(a, b)$
 $c_4 : \text{edge}(b, c)$
 $c_5 : \text{edge}(b, d)$
 $c_6 : \text{edge}(c, d)$

In Figure 2.1 we can see the SLD-Resolution tree for the query $\leftarrow \text{edge}(a, d)$ regarding the program presented in Example 2.1.6. Each edge contains the clause of the program used to derive. We omitted the substitutions θ for each derivation in order to make the figure clear. The nodes with a square represent the solution, while nodes without squares represent a failure. For instance, the substitution $\theta = \{A/b\}$ was applied to $\text{edge}(a, A), \text{path}(A, d)$ in order to derive the clause $\text{path}(b, d)$ with atom c_3 .

We say that a logic program P entails an atom a , denoted by $P \models a$, if and only if the atom a is true in the least Herbrand model of P . For the Example 2.1.6, we say that the knowledge in the program entails $\text{path}(a, d)$, which is the query, and also entails other possible paths in the graph such as $\text{path}(a, b)$ and $\text{path}(a, c)$.

For more details on First-Order logic and SLD-resolution we refer the readers to [22, 23].

2.1.2 Inductive Logic Programming

Inductive Logic Programming (ILP) is a subset of first-order logic and a research area of machine learning which is mainly concerned with inductive inference [11]. In

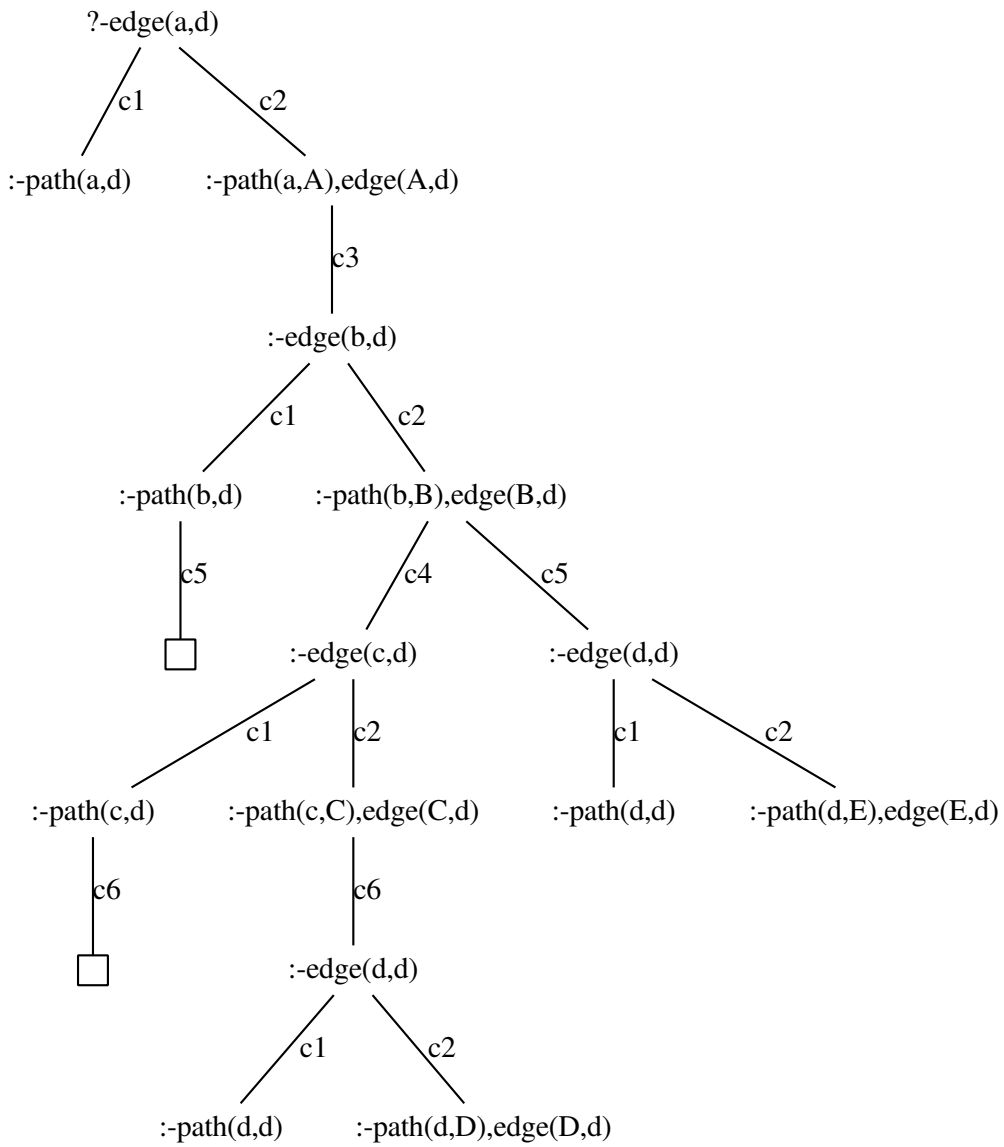


Figure 2.1: SLD-Resolution tree for the query path(a,d)

the logic context, inductive inference means reasoning from particular cases to the general case. Thus, the goal is to find a hypothesis from examples in the presence of background knowledge in order to infer the truth-value of unseen examples. This technique of inductive learning has been successfully applied to several real-world problems such as finite element mesh design [24], satellite temporal fault diagnosis [25], design of a qualitative physics [26], structure-activity prediction for drugs [27] and protein secondary structure prediction [28].

The problem of inductive learning can be defined as learning concepts. Given \mathcal{U} , a universal set of objects (or observations) and a concept \mathcal{C} as a subset of objects in \mathcal{U} , i.e. $\mathcal{C} \subseteq \mathcal{U}$, the task of learning a concept \mathcal{C} means learning how to recognize objects in the set of objects \mathcal{C} , i.e. to recognize whether $x \in \mathcal{C}$ for each $x \in \mathcal{U}$ [29]. For example, \mathcal{U} may be the set of all patients in a particular hospital, and \mathcal{C} the set of all patients having pneumonia. Similarly, \mathcal{C} may be the concept and set of all actors working in a universe \mathcal{U} of a particular movie, i.e. all people that work for a particular movie such as directors, producers, screenwriters, etc.

In order to tell whether a given object belongs to a particular concept, a hypothesis has to be learned. A hypothesis is defined as a concept description to be learned from examples. Facts are defined as object descriptions and they are part of the background knowledge. Examples are instances of the concept and can be seen as labeled facts. The examples labeled as positives are instances of the concept \mathcal{C} , while the examples labeled as negatives are not. We denote the set of examples by E , the set of positive examples by E^+ and the set of negative examples by E^- which are both subsets of $E = E^+ \cup E^-$.

In addition, if the object description satisfies the description of the concept, we say that the concept description covers the objection description. In other words, we say that the hypothesis covers the example or the example is covered by the hypothesis. This coverage relation is denoted by $\text{covers}(H, e)$ which tells if the example e is considered to belong to the concept \mathcal{C} learned by hypothesis H .

Thus, the problem of learning a concept from examples in ILP is defined as follows:

Definition 2.1.1 *Inductive learning* *Given a set E of positive E^+ and negative E^- examples of a particular concept \mathcal{C} , find a hypothesis H such that:*

- *completeness: H covers all positive examples $E^+ \in E$*
- *consistency: H does not cover any negative example $E^- \in E$*

A desired hypothesis H is the one that covers all positive examples and none of the negative ones. In this case, we say that the hypothesis H is complete and consistent with respect to the examples E . A hypothesis H is complete with respect

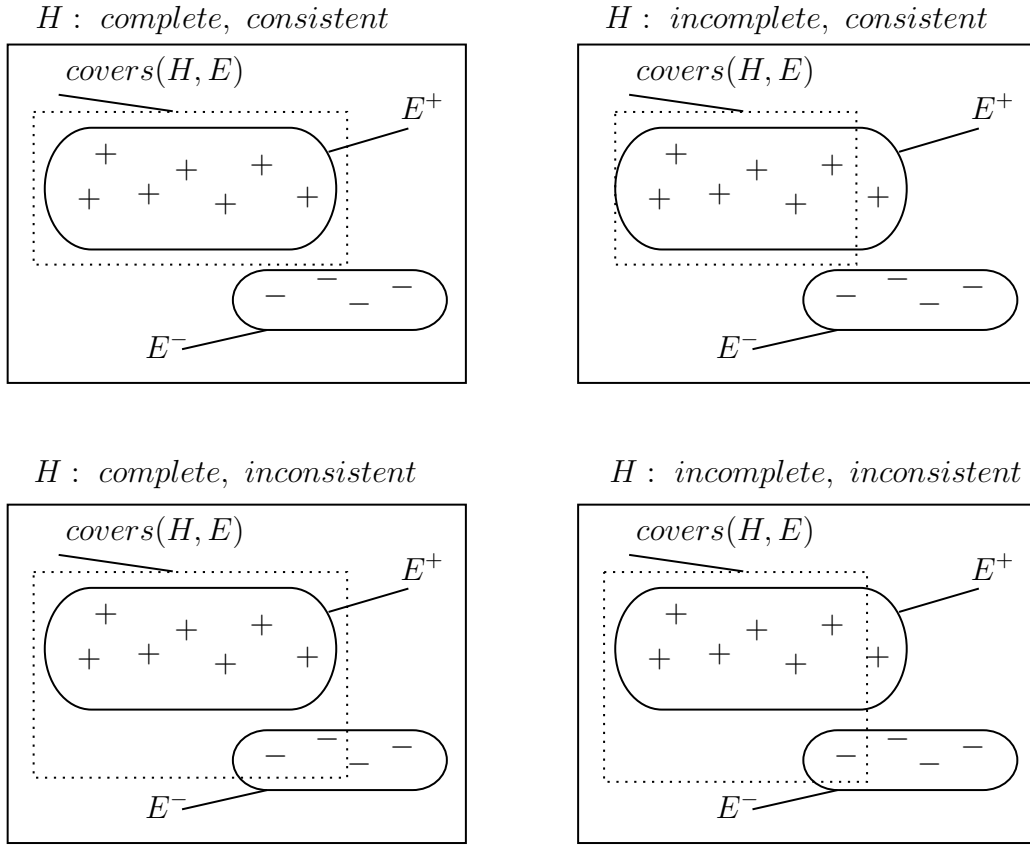


Figure 2.2: Demonstration of completeness and consistency of a hypothesis [19]

to the examples E if it covers all the positive examples, i.e. if $\text{covers}(H, E^+) = E^+$. A hypothesis H is consistent with respect to the examples E if it does not cover any negative example, i.e. if $\text{covers}(H, E^-) = \emptyset$. Figure 2.2 demonstrates possible scenarios regarding completeness and consistency of a hypothesis [19].

However, there is no guarantee that the hypothesis will satisfy both conditions of completeness and consistency. Thus, the conditions are relaxed in order to find a hypothesis as close as possible to correct (both complete and consistent). The problem is then reduced to find a hypothesis considering a criteria referred to as quality criterion. A hypothesis can also be seen as a classifier of new objects. Thus, a possible quality criterion is the accuracy of classifying unseen examples and can be measured as the percentage of correctly classified objects.

The notion of coverage in concept learning can be stated in three different formalizations [30] which are defined as follows.

Definition 2.1.2 Learning from entailment Given a clausal theory H and a clause e , H covers e iff $H \models e$.

Definition 2.1.3 Learning from interpretations Given a clausal theory H and a Herbrand interpretation e , H covers e iff e is a model of H .

Definition 2.1.4 Learning from satisfiability Given both clausal theories H and e , H covers e iff $H \wedge e \not\models \perp$.

In learning from entailment, the knowledge base, i.e. data and background knowledge, is represented as a single logic program where each example is a clause (systems usually considers ground atoms as examples).

On the other hand, learning from interpretations states examples as a set of facts (interpretations) [31]. More specifically, in the learning from interpretations setting each example e is a program that contains particular information about the example, as well as its label. The interpretation is then the set of all ground facts entailed by $e \wedge B$ where B is a program representing the background knowledge. This setting was proposed by De Raedt and Dzeroski [32].

Example 2.1.7 Suppose we have an example represented by the following program

rabbit
color(white)
likes(carrot)

and a background knowledge

eat(X) ← likes(X)

Then, the interpretation of the example is $\{rabbit, color(white), likes(carrot), eat(carrot)\}$.

Example 2.1.8 shows how a Poker game data would be represented in the learning from interpretations setting. Each example is a Herbrand interpretation and contains all facts that describe the example. The Poker data is a description of a hand of five cards which represents the poker hand categories such as full house, pair, flush, etc.

Example 2.1.8 The following example represents poker hands in the learning from interpretations setting.

Positive examples (pairs)

$\{card(7, spades), card(8, hearts), card(king, clubs),$
 $card(queen, hearts), card(7, hearts)\}$
 $\{card(4, spades), card(2, spades), card(ace, spades),$
 $card(ace, clubs), card(9, spades)\}$

and negative examples (not pairs)

$\{card(2, spades), card(5, clubs), card(ace, clubs),$
 $card(4, spades), card(queen, diamonds)\}$
 $\{card(3, hearts), card(3, clubs), card(8, clubs),$
 $card(8, spades), card(8, hearts), \}$

Learning from entailment encodes all the data, including background knowledge, in one single program. In this setting, the system has to learn from a set of examples considering all clauses and facts presented as background knowledge. This setting of learning is the most used and was introduced by Muggleton [20]. Example 2.1.9 shows how the same Poker data would be represented in the learning from entailment setting.

Example 2.1.9 *The following example represents poker hands in the learning from entailment setting.*

Ground facts

card(hand1, 7, spades)
card(hand1, 8, hearts)
card(hand1, king, clubs)
card(hand1, queen, hearts)
card(hand1, 7, hearts)
card(hand2, 4, spades)
card(hand2, 2, spades)
card(hand2, ace, spades)
card(hand2, ace, clubs)
card(hand2, 9, spades)
card(hand3, 2, spades)
card(hand3, 5, clubs)
card(hand3, ace, clubs)
card(hand3, 4, spades)
card(hand3, queen, diamonds)
card(hand4, 3, hearts)
card(hand4, 3, clubs)
card(hand4, 8, clubs)
card(hand4, 8, spades)
card(hand4, 8, hearts)

positive examples (pairs)

pair(hand1)
pair(hand2)

and negative examples (not pairs)

pair(hand3)
pair(hand4)

Whereas learning from entailment and learning from interpretations are well-known in the literature and used in practice, learning from satisfiability, introduced

by Wrobel and Dzeroski [33], is the hardest one of the three settings and is rarely used due to its computational cost. We refer the reader to [30] for further details regarding the formalizations of concept learning.

Example 2.1.10 *The following example represents parental relation among people of a particular family. Two positive and two negative examples are given as training examples. Seven facts are given as background knowledge.*

Table 2.1: Example of an ILP problem. The idea is to learn the predicate *daughter*. Positive examples are denoted by \oplus and negative by \ominus

Training example		Background knowledge	
<i>daughter(mary, ann)</i>	\oplus	<i>parent(ann, mary)</i>	<i>female(ann)</i>
<i>daughter(eve, tom)</i>	\oplus	<i>parent(ann, tom)</i>	<i>female(mary)</i>
<i>daughter(tom, ann)</i>	\ominus	<i>parent(tom, eve)</i>	<i>female(eve)</i>
<i>daughter(eve, ann)</i>	\ominus	<i>parent(tom, ian)</i>	

We illustrate the ILP task on an example of learning a particular relation from a family data. The example is given in Example 2.1.10. The target predicate *daughter*(X, Y) defines the daughter relation which states that a person X is daughter of a person Y . The background knowledge contains facts about the family which includes predicates as *female*(X) and *parent*(X, Y). The data provides two positive and two negative examples of the target relation *daughter*.

From these examples, it is expected that the ILP system learns the following rule of the target relation:

$$daughter(X, Y) \leftarrow female(X), parent(Y, X)$$

This rule says that a person X is daughter of Y if X is female and Y is parent of X . Moreover, this hypothesis is consistent and complete with respect to the background knowledge and the training examples since it covers all positive examples and none of the negative examples. For example, when substituting the variables X and Y , we see that *mary* is daughter of *ann*, i.e. $daughter(mary, ann) \leftarrow female(mary), parent(ann, mary)$ with respect to the background knowledge. On the other hand, the negative examples are not covered, as we expect. For example, we see that *tom* is not daughter of *ann* because *tom* is not even a female.

Learning from entailment is the most frequently setting used in ILP learning systems. A vast majority of systems uses it, e.g. FOIL [34], PROGOL [35], SRT [36] and FORS [37]. The learning from interpretations setting is also implemented in some systems, e.g. TILDE [38], SLIPCOVER [39] and LLPAD [40].

2.1.3 Statistical Relational Learning

One of the challenges in artificial intelligence is the combination of relational learning and reasoning under uncertainty. The traditional approach in relational learning deals with deterministic data: examples and facts are either true or false; and reasoning proves or fails to prove a particular query. In real-world problems, data have noise and are generally uncertain. Therefore, ILP systems do not formulate good models under this circumstance since they do not consider probability of data.

A variety of such approaches and learning techniques have been developed in the field known as Statistical Relational Learning (SRL) [41], Probabilistic Logic Learning (PLL) [42] or Probabilistic Inductive Logic Programming (PILP) [43]. Many variants of this field have been studied differing in the logical and in the probabilistic language.

One variant is the Probabilistic Logic Programming (PLP), which is a family of languages that are usually based on Sato’s distribution semantics [44], a well-known semantics for probabilistic logics which specifies a probability distribution over possible worlds [45]. Some examples of languages based on Prolog [46], with subtle differences, that uses this semantics are PRISM [47, 48], ICL [49, 50], LPAD [51] and ProbLog [52].

Programs consists of a set R of definite clauses (also called rules) and a set F of ground facts f_i . Each ground fact is labeled with a probability p_i and is written as $p_i :: f_i$. These labeled ground facts are called probabilistic facts and the random variables correspond directly to them. The random variables are true with probability p_i and false with probability $1 - p_i$. Assuming all these random variables are independent, the probability of a program F' is given by:

$$P_F(F') = \prod_{f_i \in F'} p_i \cdot \prod_{f_i \in F \setminus F'} (1 - p_i) \quad (2.1)$$

The set of ground facts $F' \subseteq F$ corresponds to the random variables which were assigned true values, i.e. ground facts that are true. For instance, consider the truth value assignment $burglary = true$, $earthquake = false$, $hears_alarm(mary) = true$, $hears_alarm(john) = true$ of the program given in Example 2.1.11. The set F' corresponds to the set of facts $\{burglary, hears_alarm(mary)\}$. Thus, the probability of this program F' is $0.1 \cdot (1 - 0.2) \cdot 0.7 \cdot (1 - 0.6)$. A possible world is then the set of rules added to the set of facts F' .

Example 2.1.11 *Example of a ProbLog program inspired by the alarm Bayesian network [53].*

```
0.1::burglary.      0.7::hears_alarm(mary).
0.2::earthquake.  0.4::hears_alarm(john).
```

```

alarm :- earthquake.
alarm :- burglary.
calls(X) :- alarm,hears_alarm(X).
call :- calls(X).

```

ProbLog is strongly related to the other mentioned PLP languages [54]. LPAD mainly differs by allowing disjunctions in the head of clauses where each disjunct is annotated with a probability. Example 2.1.12 shows a clause in LPAD language.

Example 2.1.12 *LPAD*

$$(heads(Coin) : 0.5) \vee (tails(Coin) : 0.5) \leftarrow toss(Coin), \neg biased(Coin)$$

In order to learn probabilistic rules, the algorithm ProbFOIL [55, 56] was developed. It combines the principles of the rule learner FOIL with ProbLog. The probabilistic rules are learned from a set of probabilistic examples and a background knowledge in the form of a ProbLog program.

The algorithm LLPAD [40] learns a LPAD program from interpretations. The algorithm finds all the disjunctive clauses that are true in all interpretations from a given set of examples in the form $(I, Pr(I))$ where I is an interpretation and $Pr(I)$ is its probability.

Another well-known variant are the Probabilistic Graphical Models (PGM), which use a graph-based representation to encode complex joint probability distributions in a compact way [57]. They combine uncertainty with logical structure to compactly represent real-world complex problems. In this graph-based representation, the nodes correspond to random variables in a domain, and the edges correspond to direct probabilistic interactions between nodes.

A surge of interest in PGM due to its flexibility and effectively learning and performing inference in large networks has led to the development of a number of approaches. Two prominent approaches are Markov Logic Network and Relational Dependency Network.

Markov Logic Network (MLN) [13] is a simple graph representation that combines probability and first-order logic. Each formula has a weight attached which reflects the difference in log probability of a possible world that satisfies the formula and one that does not.

It provides a template for constructing Markov networks with one node for each ground predicate given a finite set of constants. The distribution probability is induced by the following way:

$$P(X = x) = \frac{1}{Z} \exp\left(\sum_{i=1}^F w_i n_i(x)\right) \quad (2.2)$$

Where F is the number of formulas, w_i is the weight of the respective clause, $n_i(x)$ is the number of true groundings of F_i in possible world x and Z is a normalization constant.

Algorithms have been proposed in order to learn the weights associated with formulas (parameter learning) [58], and also in order to learn the formulas themselves (structure learning) [59].

Example 2.1.13 *Example of a MLN. $Fr/2$ is short for $Friends/2$, $Sm/1$ for $Smokes/1$, and $Ca/1$ for $Cancer/1$. The first formula states that friends of friends are friends. Second formula states that smoking causes cancer.*

Table 2.2: An example of MLN with two formulas and their respective weights

Weight	Clausal Form
0.7	$\neg Fr(X, Y) \vee \neg Fr(Y, Z) \vee Fr(X, Z)$
1.5	$\neg Sm(X) \vee Ca(X)$

Since Relational Dependency Networks is the PGM used in the RDN-Boost algorithm, we will introduce it in details in Section 2.4.1.

2.2 Theory Revision

As explained in Section 2.1.2, learning FOL theories from a set of examples and background knowledge is a process known as ILP. Given the background knowledge, represented as definite clauses, and a set of examples, represented as logical facts, an ILP learner derives a hypothesis in the form of a logic program that covers as many as possible positive examples and avoids covering negative examples. For the reason that such systems start from an empty initial hypothesis, we say that they learn from scratch.

Nevertheless, an incomplete or partially correct theory may exist and the system may take advantage of it as a starting point in order to improve it, instead of discarding it and learning a new theory from scratch. Such incomplete or partially correct theory may exist due to different reasons: (1) it has been proposed by a domain expert who has only partial knowledge about the domain; (2) it has become incomplete due to new data available (outdated); (3) it has come from a different yet related domain through transference; etc.

In such cases, the theory may contain important information that can still explain some data and because of that should not be discarded. Thus, it is desirable to use the given theory as a starting point in the learning process and take advantage of its existing information. These important considerations have contributed to the development of Theory Revision systems [60-62].

2.2.1 Definition

Theory revision is the process of repairing incorrect or incomplete theories from a set of examples. It is a sub-task of a general problem known as theory refinement. A theory revision task consists of proposing modifications to the theory resulting in changes in the set of answers, i.e covering missing answers or fixing incorrect answers made by the theory. On the other hand, theory restructuring task, which is another sub-task of theory refinement, does not change the set of answers. Figure 2.3 shows an overview of the refinement tasks divided into revision and restructuring. Restructuring is concerned with improving performance and understandability of a theory.

Theory revision is then defined as follows [18]:

Definition 2.2.1 *The task of theory revision is*

Given

- *an initial theory H*
- *a background knowledge BK*
- *a set of positive and negative examples E^+ and E^- composing the set of examples E*

Find

- *a revised theory H'*
- *that covers all the positive examples (completeness), $BK \wedge H' \models E^+$*
- *and none of the negative examples (consistency), $BK \wedge H' \not\models E^-$*
- *and obeys a minimality criteria*

The goal of a theory revision task is to use a given initial theory H , a background knowledge BK written as definite clauses and a set of positive E^+ and negative E^- examples written as ground definite clauses in order to find a revised theory H' that covers all the positive examples and none of the negative examples, and also obeys a minimality criteria which requires minimal revisions of the theory. However, it is not always possible to find a complete and consistent theory, thus theory revision systems find theories as close as possible to be complete and consistent. Basically, theory revision systems differ from learning from scratch by starting with a given theory as starting point and attending to minimality criteria.

Usually, theory revision is applied when new data have become available or when an imprecise theory learned from scratch needs to be improved. It commonly performs revision in four main steps:

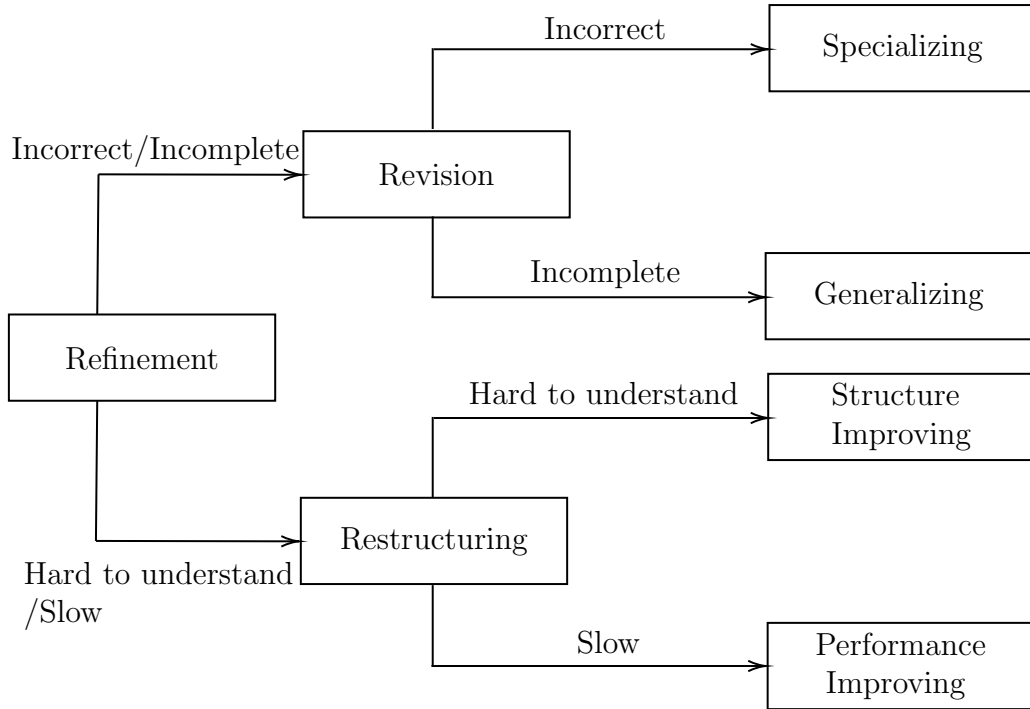


Figure 2.3: Taxonomy of theory refinement tasks [18]

1. Finding incorrectly classified examples by the theory.
2. Searching for clauses and literals responsible for incorrect classifications of those examples.
3. Proposing modifications to such points through applying possible operations.
4. Scoring proposed revisions and selecting the best.

These points in the theory responsible for incorrect classifications are called revision points and the possible operations are called revision operators. Next, we briefly explain these concepts.

2.2.2 Revision Points

The initial theory is assumed to be partially correct and thus, only some points are responsible for misclassifications or bad predictions of examples. These points are called revision points and are detected according to misclassified examples. When positive examples are not covered (i.e. false negatives), the theory is too specific. On the other hand, when negative examples are covered (i.e. false positives), the theory is too general. Usually, revision points are divided into two types:

- **Specialization revision points**

Clauses in the theory responsible for successfully proving negative examples,

i.e. false positives. It indicates that the theory is too general and, therefore, needs to be specialized to avoid proving negative examples.

- **Generalization revision points**

Literals and clauses in the theory responsible for the failure of proving positive examples, i.e. false negatives. It indicates that the theory is too specific and, therefore, needs to be generalized to allow positive examples to become provable.

Often, several clauses can be part of the proof of the negative examples. In addition, several clauses could be generalized in order to make positive examples covered. All such clauses can be marked as revision points together with literals potentially responsible for misclassification. In the case of specialization revision points, such clauses and literals can be obtained from the refutation path in the SLD-tree, while generalization revision points can be obtained from a failure path. After finding such revision points, the system proposes modifications by applying the revision operators.

2.2.3 Revision Operators

As mentioned, only some points are responsible for misclassifications or bad predictions. Therefore, a theory revision system needs only to propose modifications for such points instead of discarding the initial theory or proposing modifications for all its clauses [62]. Revision operators are responsible for proposing modifications at each revision point and the type of the revision point determines which revision operator will be applied. Commonly, two types of revision operators are considered: (1) generalization operators; and (2) specialization operators [18]. Generalization operators can be used to handle false negatives, i.e. applied to generalization points, while specialization operators can be used to remove false positives, i.e. applied to specialization points.

The most common used operators are the following [62] generalization operators,

- **Delete-antecedents**

This operator erases literals marked as revision points from clauses that prevent proving positive examples.

- **Add-rule**

This operator inserts new clauses into the theory, either from existing clauses or from scratch, in order to prove positive examples.

and the following specialization operators,

- **Delete-rule**

This operator removes a clause responsible for proving a negative example.

- **Add-antecedents**

This operator inserts literals to a clause marked as revision point in order to avoid proving negative examples.

We refer the reader to [18] for further information about revision operators.

2.2.4 FORTE Algorithm

The first step of the theory revision process finds misclassified examples by applying the SLD-Resolution where the root is an example $e \in E$. In the SLD-tree, nodes are objective clauses while edges are composed of a θ -substitution and a clause from BK or H' . A path is a refutation path if its leaf is an empty clause represented by a square and a failure path otherwise. As mentioned earlier, revision points can be obtained from refutation or failure paths which is the second step. The size of the search grows with the number of misclassified examples, number of clauses and literals responsible for misclassification and the size of the knowledge base used to propose modifications. The third step consists of applying the matching revision operators at each revision point in order to generate new possible theories. Finally, the system keeps the theory that yields the best performance, given an arbitrary metric, among the generated revised theories. The algorithm repeats this process until no revision improves the theory. An overview of the FORTE [60], a theory revision algorithm, can be seen in Algorithm 1.

Algorithm 1 FORTE Algorithm

```

repeat
  generate revision points
  sort revision points by their potential
  for each revision point do
    generate revisions
    update best revision found
  until potential of next revision point is less than the score of the current best
  revision
  if best revision improves the theory then
    implement best revision
  end if
until no revision improves the theory

```

Potential is defined as the number of misclassified examples that could be correctly classified. As can be seen, FORTE performs hill-climbing search through a space of revision operators in order to find a consistent revised theory [63].

2.3 Transfer Learning

Traditional machine learning algorithms work with the assumption that training data and future data used for prediction must be in the same feature space and must have the same distribution. However, this assumption may not hold in real-world scenarios and traditional learned models may fail in these tasks. Then, when the test distribution changes w.r.t. to the training data, the algorithms need to relearn from scratch using the newly collected data and use these rebuilt models for new predictions. The test distribution may change, for example, due to easily outdated data.

Arguably, a more efficient solution would be to adapt the previously learned model to the new distribution of examples. Another situation that may benefit from adaptability is when collecting data is quite expensive or even impossible for a particular domain. However, it could be the case that, while we do not have sufficient data for a specific domain, we may have plenty of data for a similar domain. While obtaining real-world measures would be extremely expensive, generating data from simulations, which have a different distribution comparing to reality, could be easier (e.g. a physics engine mimicking movements of a robot).

Transfer learning [6] addresses the problem of lacking data by allowing that domains used in training and testing be different. The advantage is to exploit the knowledge learned in a source domain to improve the performance of a related target domain. An example might be using the knowledge obtained to recognize a specific kind of object to help to recognize a similar object, or equivalently, using knowledge learned from Spanish and apply it to learn Portuguese. Indeed, transfer learning is motivated by the fact that humans are able to take the knowledge learned in a specific domain and apply it to a completely different domain. The difference between traditional learning and transfer learning is that traditional learning tries to learn a task from scratch regarding one specific domain, while transfer learning tries to transfer knowledge learned from a previous source task to a target task.

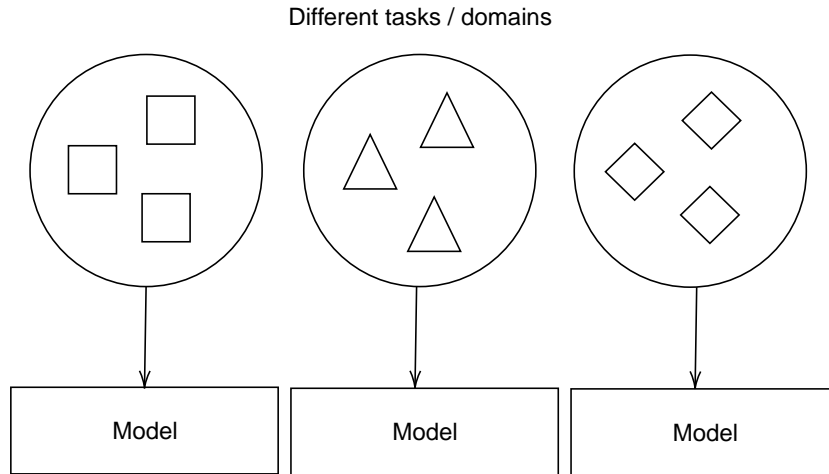


Figure 2.4: Traditional machine learning setup

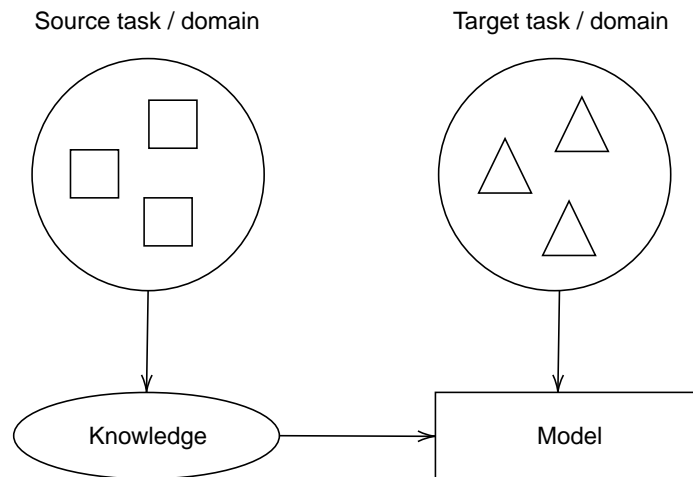


Figure 2.5: Transfer learning setup

To sum up, in the traditional machine learning scenario depicted in Figure 2.4, a model is intended to be trained for a given task and domain and, thus, it is assumed that the system is provided with labeled data for the same task and domain. Then, a model is trained on this data and expected to perform reasonably well on unseen data of the same task and domain. The transfer learning scenario, depicted in Figure 2.5, differs by allowing us to consider data from different tasks and domains known as source and target task/domain. In this scenario, where we do not have sufficient labeled data for the target task or domain, the knowledge gained in solving a source task in a source domain is applied to obtain a model to this particular target task and target domain.

2.3.1 Definition

A definition of transfer learning was presented by [6] as the following: a domain \mathcal{D} is consisted by a feature space \mathcal{X} and a marginal probability distribution $P(X)$ where $X = \{x_1, \dots, x_n\} \in \mathcal{X}$. Considering the problem of document classification as an example, then \mathcal{X} is the space of all term vectors, x_i is the i -th term of a vector of a given document and X is a particular sample. Given a domain $\mathcal{D} = \{\mathcal{X}, P(X)\}$, a task \mathcal{T} consists of a label space \mathcal{Y} and a not observed conditional probability distribution $P(Y|X)$ which could be learned from training data. In the document classification example, \mathcal{Y} is the set of all labels which is either True or False. Finally, given a source domain \mathcal{D}_S and a source task \mathcal{T}_S , also a target domain \mathcal{D}_T and a target task \mathcal{T}_T , the purpose of transfer learning is to help to learn the target conditional probability distribution $P(Y_T|X_T)$ in \mathcal{D}_T using the knowledge obtained from \mathcal{D}_S and \mathcal{T}_S , where $\mathcal{D}_S \neq \mathcal{D}_T$, or $\mathcal{T}_S \neq \mathcal{T}_T$.

If two domains are different, they have either a different feature space or different marginal probability distributions due to the definition of the domain as a pair $\mathcal{D} = \{\mathcal{X}, P(X)\}$. Thus, the condition $\mathcal{D}_S \neq \mathcal{D}_T$ implies that $\mathcal{X}_S \neq \mathcal{X}_T$ or $P_S(X) \neq P_T(X)$. In document classification, a different feature space may correspond to domains of two different languages and different probability distribution may correspond to domains in the same language but about different topics. Similarly, for a definition of task as a pair $\mathcal{T} = \{\mathcal{Y}, P(Y|X)\}$, the condition $\mathcal{T}_S \neq \mathcal{T}_T$ implies that $\mathcal{Y}_S \neq \mathcal{Y}_T$ or $P(Y_S|X_S) \neq P(Y_T|X_T)$.

Definition 2.3.1 *Transfer learning.* Given a source domain \mathcal{D}_S , a source task \mathcal{T}_S , a target domain \mathcal{D}_T and a target task \mathcal{T}_T , transfer learning aims to help learning the target conditional probability distribution $P_T(Y_T|X_T)$ in \mathcal{D}_T with the knowledge extracted from \mathcal{D}_S and \mathcal{T}_S , where $\mathcal{D}_S \neq \mathcal{D}_T$, or $\mathcal{T}_S \neq \mathcal{T}_T$.

2.3.2 Research issues

There are three main important research issues regarding transfer learning: (1) what to transfer, (2) how to transfer, and (3) when to transfer [6].

What to transfer decides what kind of knowledge to transfer between domains or tasks. Some knowledge may be common between both the domains so that it could help improving performance in a target domain. For relational domains, the knowledge to be transferred may be represented by the structure of a theory and a mapping from source predicates to target predicates must be found in order to found which clauses to transfer to the target domain.

How to transfer concerns how to actually perform transfer needs to be considered and learning algorithms must be developed to accomplish this process. Most of the work focus on these two issues and, thus, algorithms and techniques are developed considering what to transfer across domains and how to proceed with the transference.

When to transfer corresponds to answering when transferring should be done or not. In some situations, referred to as negative transfer, transference may hurt the learning performance in the target domain resulting in worse accuracy than to not transfer at all. Knowing when transference should not be done is also an interesting issue in order to avoid struggling in a transference that would lead to an unsuccessful or worse result.

2.3.3 Scenarios

In transfer learning, the source and target conditions may vary in different ways, which is illustrated in 4 different scenarios [64]. The first two scenarios occur when there is difference between the source and target domains, i.e. $\mathcal{D}_S \neq \mathcal{D}_T$, while the last two scenarios occur when there is difference between the source and target tasks, i.e. $\mathcal{T}_S \neq \mathcal{T}_T$.

1. $\mathcal{X}_S \neq \mathcal{X}_T$. The feature spaces of the source and target domains are different, e.g. the documents are written in two different languages or students' course grades in two different academic specializations.
2. $P(X_S) \neq P(X_T)$. The marginal probability distributions of the source and target domains are different, e.g. the documents describe different topics but in the same language, or food consumption in two different diets.
3. $\mathcal{Y}_S \neq \mathcal{Y}_T$. The label spaces of the source and target tasks are different. This scenario usually occurs with scenario 4 since it is very rare to exist a case where two different tasks have different labels spaces, but exactly the same conditional probability distributions.
4. $P(Y_S|X_S) \neq P(Y_T|X_T)$. The conditional probability distributions of the source and target tasks are different, e.g. the approval of a particular product in two different states, or the distribution of political positions in two different countries.

The setting when the target task is different from source task, no matter if the source and target domains are the same or not, is known as inductive transfer learning. In contrast, when source and target tasks are the same, the setting is

known as transductive transfer learning. Such transfer learning settings we will discuss below.

2.3.4 Taxonomy

Firstly, it is important to point the difference between transductive and inductive learning in the context of traditional learning approach. Inductive learning attempts to discover rules or generalizations from collected samples. In other words, it returns a function learned from training samples in order to make generalizations that can help classifying unseen instances in the future. It discards information potentially conveyed by the unlabeled instances in the process of training. Conversely, the idea of transductive is to use both labeled and unlabeled training data to classify only the unlabeled data in one step. Since transductive learning does not learn a model, the training data has to be used whenever new instances have to be classified. [65, 66].

Different from the traditional learning approach, transfer learning considers data from both source and target domain. We now categorize transfer learning into three different settings by adapting the taxonomy presented in [6, 64]. The different settings are presented and described in Table 2.3 and Figure 2.6.

The transductive transfer learning, which term was first proposed by ARNOLD *et al.* [67], requires that both source and target tasks be the same and labeled data is presented only in the source domain. It can be divided into two cases: (1) features spaces between source and target domains are different, i.e. the first scenario where $\mathcal{X}_S \neq \mathcal{X}_T$, and (2) feature spaces are the same, i.e. $\mathcal{X}_S = \mathcal{X}_T$, but the marginal probability distributions are different, i.e. the second scenario where $P(X_S) \neq P(X_T)$. These cases are related to domain adaption [68], sample selection bias [69] or covariance shift [70]. The transductive transfer learning setting is defined below.

Definition 2.3.2 *Transductive transfer learning.* Given a source domain \mathcal{D}_S , a source task \mathcal{T}_S , a target domain \mathcal{D}_T and a target task \mathcal{T}_T , transductive transfer learning aims to help learning the target conditional probability distribution $P_T(Y_T|X_T)$ in \mathcal{D}_T with the knowledge extracted from \mathcal{D}_S and \mathcal{T}_S , where $\mathcal{D}_S \neq \mathcal{D}_T$ and $\mathcal{T}_S = \mathcal{T}_T$, and also with some unlabeled target domain data available to use in training time.

The inductive transfer learning setting requires labeled data in target domain and that source and target tasks are different, no matter if the source and target domain are the same or not. This setting has two different cases: (1) labeled data in source domain are available and (2) labeled data in source domain are not available. The sequential transfer learning, a term introduced by RUDER [64] in order to highlight the difference to multi-task learning, consists of two stages: a pretraining phase and

an adaptation phase. The model is trained in a source task in the pretraining phase and then the knowledge of the model is transferred to a target task in the adaptation phase [64]. This differs from multi-task learning [71] where source and target tasks are learned at the same time. Multi-task learning specifically requires labeled data in source domain, i.e. the first case. An instance of the sequential transfer learning is the self-taught learning [72] which does not count with labeled data in source domain, i.e. the second case. The inductive transfer learning setting is defined as follows.

Definition 2.3.3 *Inductive transfer learning.* Given a source domain \mathcal{D}_S , a source task \mathcal{T}_S , a target domain \mathcal{D}_T and a target task \mathcal{T}_T , inductive transfer learning aims to help learning the target conditional probability distribution $P_T(Y_T|X_T)$ in \mathcal{D}_T with the knowledge extracted from \mathcal{D}_S and \mathcal{T}_S , where $\mathcal{T}_S \neq \mathcal{T}_T$.

The unsupervised transfer learning setting deals with the absence of labeled data in both source and target domain. It focuses on solving unsupervised learning tasks in target domain by using knowledge from source domain. Such tasks are (1) clustering, such as the self-taught clustering [73], and (2) dimensionality reduction, such as the transferred discriminative analysis [74].

Definition 2.3.4 *Unsupervised transfer learning.* Given a source domain \mathcal{D}_S , a source task \mathcal{T}_S , a target domain \mathcal{D}_T and a target task \mathcal{T}_T , unsupervised transfer learning aims to help learning the target predictive function $f_T(\cdot)$ in \mathcal{D}_T with the knowledge extracted from \mathcal{D}_S and \mathcal{T}_S , where $\mathcal{T}_S \neq \mathcal{T}_T$ and both \mathcal{Y}_S and \mathcal{Y}_T are not observable.

Table 2.3: Different settings of transfer learning

Setting	Source domain labels	Target domain labels
Inductive transfer learning	Available/Unavailable	Available
Transductive transfer learning	Available	Unavailable
Unsupervised transfer learning	Unavailable	Unavailable

2.3.5 Approaches

There exist different approaches in the literature that performs transfer learning by transferring different kinds of knowledge from source to target domain. PAN and YANG [6] described four distinct cases which includes **instance-transfer** [7, 75], **feature-representation-transfer** [72, 76] and **parameter-transfer** [77, 78]. We refer the reader to their work for more information. The case referred to as **relational-knowledge-transfer** deals with transfer learning for relational data

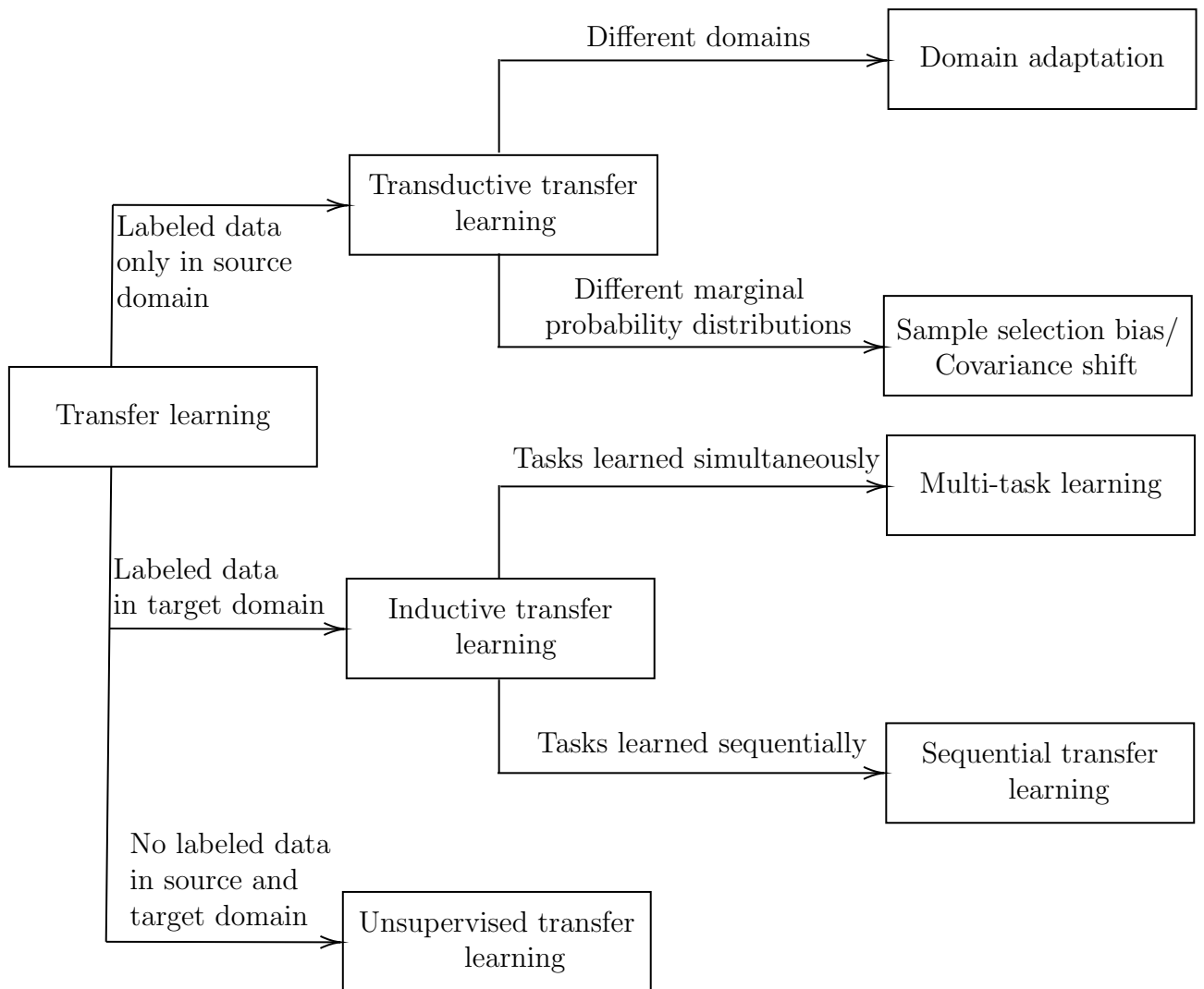


Figure 2.6: Taxonomy of transfer learning settings [6, 64]

scenario and is studied in the inductive transfer learning setting. Most of the current work conducted in transfer learning assumes that the source and target domains are related, i.e. there exists some relationship between both feature spaces [6]. In the relational data scenario, it is assumed that the source and target data may share similar relationships. If two domains are related to each other, there may exist two similar relationships that connect entities in a domain and thus a mapping for these relationships may be found. For example, a professor in a university domain can be considered similar to a director in a cinematographic domain because they play similar roles by teaching/leading students and actors. In addition, the relationship between a professor and a student, as well as between a director and an actor can be considered similar. Consequently, other relationships as "professor publishing a paper" and a director directing a film can also be considered similar. More examples would be the relationship of a professor teaching students which is similar to a football coach teaching football players, also a team playing a specific sport which is similar to a company belonging to a specific economic sector. This case is the one related to this work and the algorithms proposed for this problem will be introduced in Section 2.5.

2.4 RDN-Boost

The following section introduces the base algorithm used for this work, namely RDN-Boost. Whereas current learning approaches for relational dependency networks learn only a single relational probability tree per predicate, RDN-Boost learns a set of relational regression trees using gradient-based boosting. In this section, we will first review the concept of relational dependency networks. Second, we review the functional gradient boosting method. Then, we describe how relational regression trees are learned. Finally, we describe the RDN-Boost algorithm.

2.4.1 Relational Dependency Networks

Bayesian networks (BN) are directed probabilistic models that have been used for learning and inference. A disadvantage of this graphic model is that it cannot capture cyclic dependencies that might occur in data. Also, learning the structure of BNs is a hard problem since the inference of BNs is NP-hard. HECKERMAN *et al.* [79] introduced the Dependency networks (DN) which one of its advantages is allowing cyclic dependencies by accepting bi-directional relationship between random variables. Exploiting these cyclic dependencies has been shown to significantly improve accuracy compared to graphic models that cannot model cyclic dependencies. DNs approximate the joint distribution as a product of individual conditional proba-

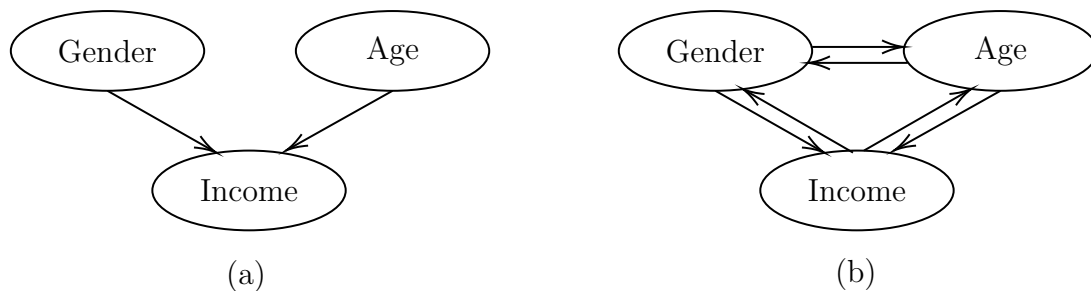


Figure 2.7: (a) An example of Bayesian network. (b) The corresponding dependency network example

bility distributions (CPD) learned independently, i.e. $P(y_1, \dots, y_n | \mathbf{X}) \approx \prod_i P(y_i | \mathbf{X})$. The main distinction of DNs to other graphic models such as Bayesian networks and Markov networks is that DNs are an approximate model. HECKERMAN *et al.* [79] introduced a Gibbs sampling method called *ordered pseudo-Gibbs sampler* that can be used to recover the full joint distribution of DNs.

DNs are represented by directed graphs $G = (V, E)$ which allows bi-directional edges between variables. This representation combines the characteristics of both directed and undirected models. In DNs, the edges between variables denotes the relationship between them and the conditional probability distribution $P(V_i | \mathbf{Pa}(V_i))$ that gives the probability of the feature v_i given its parent $\mathbf{Pa}(V_i)$. Figure 2.7 shows networks that describe the demographic characteristics of visitors to a website [79]. In Figure 2.7a, Income is predictive of Age and Gender, while in Figure 2.7b there are cyclic dependencies among the variables.

On the other hand, Relational Dependency Networks (RDN) [80] extends DNs to work with relational data. This is similarly done in the way RBNs [81] extend Bayesian networks and RMNs [82] extend Markov networks. RDNs approximate the joint distribution as a product of conditional probability distributions over ground atoms. The fact that RDNs are approximate models is the primary difference between RDNs and other directed SRL models [17, 80]. RDNs consist of a set of predicates that can be grounded given the instantiation of variables. In addition, similarly to DNs, each predicate Y_i is associated with a CPD $P(Y_i | \mathbf{Pa}(Y_i))$. Figure 2.8 presents an example of RDN for a university domain. As can be seen, the objects *professor*, *student* and *course* interact with each other through the relations *takes* and *taughtby*. The edges in the graph denote probabilistic dependencies between the predicates.

Since RDNs can be represented as a set of conditional distributions, learning RDNs corresponds to learning these distributions independently. NEVILLE *et al.* [83] used relational probability trees (RPT) and relational Bayesian classifiers (RBC) [84] to learn these distributions. Of the two, RPTs have yielded better results and

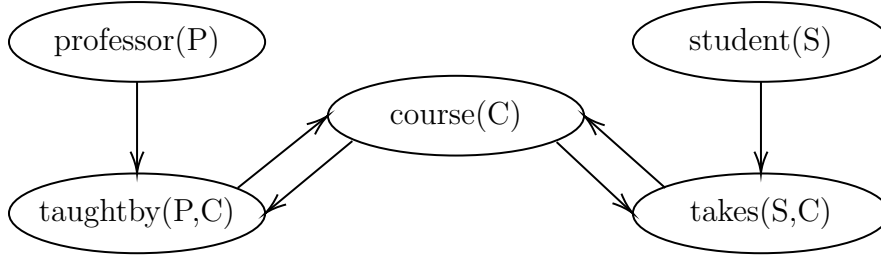


Figure 2.8: Example of RDN for a university domain

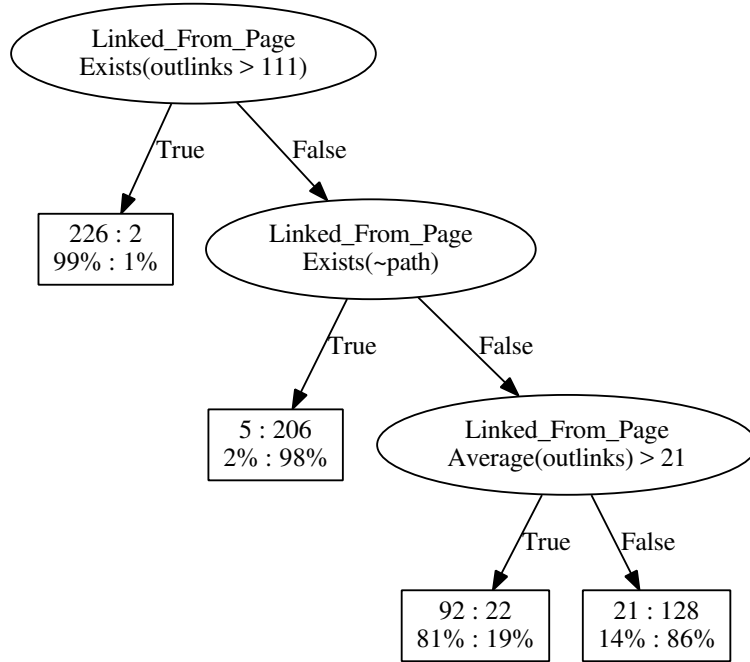


Figure 2.9: Example of RPT

became more popular to represent CPDs in RDNs. An example of RPT is presented in Figure 2.9. The leaves contain the probability distributions of the target label and the proportion of positive and negative examples that reached the given leaf node. RPTs are learned similarly to relational regression trees which will be introduced in Section 2.4.3. We refer the readers to [83] for further details on RPTs.

2.4.2 Functional Gradient Boosting

Functional gradient methods were used to train conditional random fields (CRF) [85], as well as their relational extension TILDE-CRF [86]. The assumption of standard gradient ascent methods is that ψ can be parameterized as a linear function,

$$\psi = \sum \beta_i f_i \quad (2.3)$$

The assumption of functional gradient ascent is more general. Instead of assuming linear parameterization, it assumes that ψ is represented by a weighted sum of

functions,

$$\psi_m = \psi_0 + \Delta_1 + \dots + \Delta_m \quad (2.4)$$

Each functional gradient at episode m is given by,

$$\psi_m = \eta_m E_{\mathbf{x},y} \left[\frac{\partial}{\partial \psi_{m-1}} \log P(y|\mathbf{x}; \psi_{m-1}) \right] \quad (2.5)$$

Where η_m is the learning rate, y denotes the grounding of the predicate and \mathbf{x} denotes the sets of groundings, i.e. the facts. It is important to mention that when we learn a full RDN, each predicate becomes the query predicate successively. DIETTERICH *et al.* [85] pointed out that the expectation $E_{\mathbf{x},y}$ cannot be computed since the joint distribution $P(\mathbf{x}, y)$ is unknown.

However, since we have training examples sampled from this joint distribution, we can compute the functional gradients at each training example in the following way,

$$\Delta_m(y_i, \mathbf{x}_i) = \frac{\partial}{\partial \psi_{m-1}} \sum_i \log P(y_i|\mathbf{x}_i; \psi_{m-1}) \quad (2.6)$$

Then, DIETTERICH *et al.* [85] suggested to train a regression function $h_m(y, \mathbf{x})$ on training examples $((\mathbf{x}_i, y_i), \Delta_m(y_i, \mathbf{x}_i))$ so that it approximates $\Delta_m(y_i, \mathbf{x}_i)$. This regression function is trained in form of regression tree h_m and fitted in order to minimize,

$$\sum_i [h_m(y_i, \mathbf{x}_i) - \Delta_m(y_i, \mathbf{x}_i)]^2 \quad (2.7)$$

In the RDN-Boost, these regression trees are relational as we will explain further. DIETTERICH *et al.* [85] pointed out that although the fitted function h_m and the desired Δ_m are not exactly the same, assuming there are enough training examples, the function will point to the same general direction, i.e. ascent in the direction of h_m will approximate the true functional gradient ascent.

The benefits of a boosting approach are to RDNs are: (1) the number of parameters grows only with the number of training episodes. Interactions among variables are introduced only as needed and the algorithm does not explicitly consider the potentially large search space; (2) the algorithm is fast and straightforward to implement. In addition, the algorithm may avoid overfitting due to the effect of combining multiple regression trees [85]; and (3) allows learning both the structure and the parameters of RDNs simultaneously, which is an attractive feature since structure learning in SRL models is computationally expensive.

2.4.3 Relational Regression Trees

As mentioned earlier, to learn RDNs, each conditional probability distribution can be represented as RPTs, as done by NEVILLE *et al.* [83]. However, following a previous work [87], NATARAJAN *et al.* [17] replaced the RPT of each distribution with a set of relational regression trees (RRT) [38] built sequentially, i.e. learning a set of gradient boosted trees instead of a single tree. This approach resulted in the algorithm called RDN-Boost which has been shown to have state-of-the-art results in learning RDNs. An example of the first RRT is provided in Figure 2.10. Several boosted trees are learned for a given relation and combined in ensemble.

The idea was to apply Friedman’s gradient boosting [88] to RDNs and represent each conditional probability distribution as a weighted sum of regression models. Specifically, each relation is represented as a set of RRTs [38]. A particular advantage of the boosting method is that it allows learning both the structure and the parameters of RDNs simultaneously. RDN-Boost uses RRTs and computes functional gradients for each training example. As explained earlier, the functional gradient starts with an initial potential ψ_0 and iteratively adds gradients Δ_i resulting after m iterations in the potential described in Equation 2.4. After these m steps, the current model will have m regression trees for a given query predicate. The regression tree learner takes weighted examples and finds a regression tree h_m that minimizes Equation 2.7. The weight of an example corresponds to the gradient presented to that example. For each tree, the probability and the gradient of an example is computed based on its groundings. Then, the gradient serves as the weight for the example at the next training step.

In RRTs, inner nodes (or test nodes) are conjunctions of literals, and a variable presented in a node cannot appear in its right subtree (i.e. variables are bounded along left-tree paths. This restriction is due to the fact that the right subtree is only relevant when the conjunction of literals fails.) [86]. A RRT is learned as the following: the learning algorithm starts with an empty tree and repeatedly searches for the best test for a node according to some splitting criterion. Then, the examples in the node are split into *success* and *failure* according to the test. The examples covered by the left path reaches the *success*, while the examples not covered reaches the *failure* (right path). The splitting criterion used by RDN-Boost, and also this work, was weighted variance on *success* and *failure*. For each split, the procedure is recursively applied further in order to obtain subtrees for the respective splits. The procedure stops if the variance in one node is small enough, the tree has reached a maximum depth defined in the procedure or has derived a maximum number of leaves. Finally, the leaves are the computed average regression values [17, 86]. An example is presented in Figure 2.10. This tree was learned for the

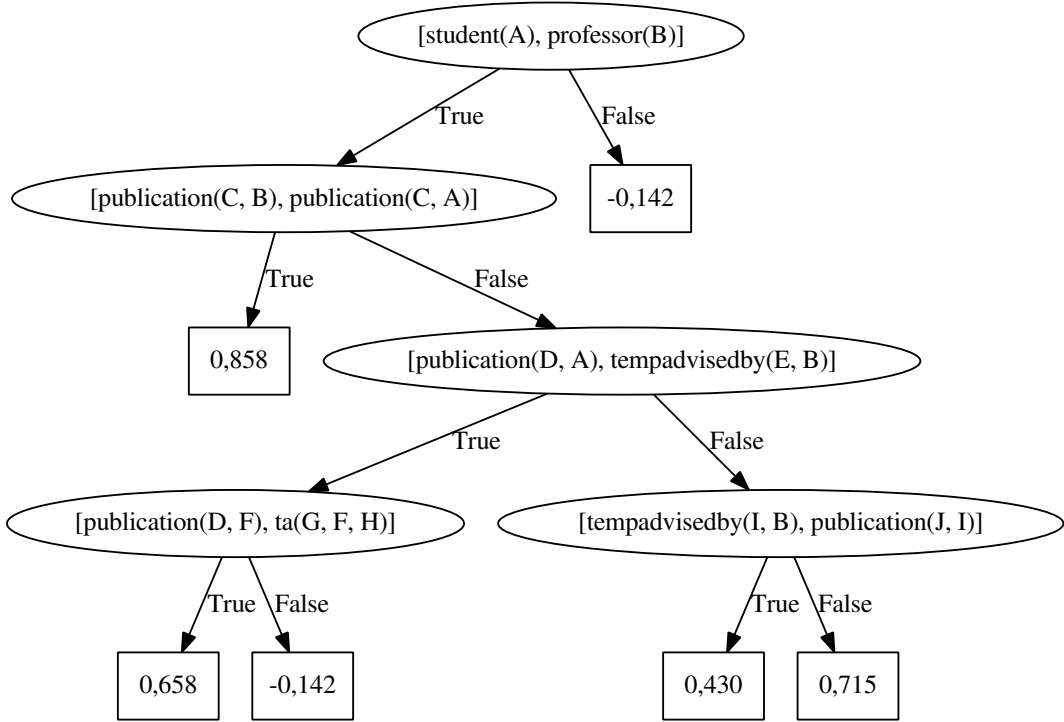


Figure 2.10: Example of RRT

query predicate *advisedby*, thus the goal is to predict if A is advised by B . In the tree, if A is a *student*, B is a *professor* and both work in the same publication ($publication(C, B), publication(C, A)$), then the regression value is 0.858. On the other hand, if the node $student(A), professor(B)$ is not satisfied, then the regression value is -0.142. Negative values indicate lower probabilities and, for this tree, the fact that A is not student or B is not a professor indicate a lower probability of A being advised by B . This regression tree learner also considers aggregation functions such as *count*, *max*, *average* in the inner nodes, however we did not consider aggregation functions in our work. For more details about aggregation functions, we refer the reader to [17].

Note that each path between the root and leaf in the regression tree can be considered as a clause in a logic program. For the Figure 2.10, the left-most path in the tree is the clause: $student(A) \wedge professor(B) \wedge publication(C, B) \wedge publication(C, A) \Rightarrow advisedby(A, B)$. The clauses are evaluated in order from left to right for a particular query and the corresponding regression value is returned. In the logical setting, multiples ground clauses can be satisfied for a particular query. In the case of relational regression trees, this is avoided by considering only the first satisfied ground clause (i.e. the first path that covered an example). This is equivalent to add a cut to the end of each clause in a logic program. Then, the second clause to be evaluated in the example if the first clause were not satisfied would be the clause made by second left-most path.

2.4.4 Learning algorithm

The algorithm RDN-Boost presented by NATARAJAN *et al.* [17] is described in Algorithm 2. The main algorithm *TreeBoostForRDNs* iterates over K predicates and for each predicate it generates the examples for the regression tree learner, which is called through *FitRelRegressTree*. The regression tree learner updates the current model F_m^k . This process is repeated for a number M of gradient steps, i.e. number of boosted trees. After m steps, the current model F_m^k will approximate the corresponding gradient for the predicate k . The initial potential F_0^1 is usually set to capture the uniform distribution. Then, ψ^k is obtained by grounding the trees where each tree determines the branch that satisfies a particular grounding and their corresponding regression values are added to the potential.

The function *GenExamples* provides the examples with their respective weights to the regression tree learner. It takes as input the predicate k , the examples and the current model F . Then, the function iterates over all examples and computes the probability and the gradient for each example. The regression values are computed for each tree based on the grounding of the examples and added to the potential. Then, the gradient is set as the weight of the example. The set of RRTs composes the structure of the conditional distribution while the set of leaves composes the parameters of the conditional distribution. In this way, the gradient boosting allows the algorithm to learn both the structure and the parameters of the RDN simultaneously.

Algorithm 2 RDN-Boost algorithm

```

function TREEBOOSTFORRDNs(Data)
  for  $1 \leq k \leq K$  do
    for  $1 \leq m \leq M$  do
       $S_k \leftarrow \text{GenExamples}(k, \text{Data}, F_{m-1}^k)$ 
       $\Delta_m(k) \leftarrow \text{FitRelRegressTree}(S_k, L)$ 
       $F_m^k \leftarrow F_{m-1}^k + \Delta_m(k)$ 
    end for
     $P(Y_k = y_k | \mathbf{Pa}(X_k)) \propto \psi^k$ 
  end for
end function
function GENEXAMPLES( $k, \text{Data}, F$ )
   $S \leftarrow \emptyset$ 
  for  $1 \leq i \leq N_k$  do
    Compute  $P(y_k^i = 1 | \mathbf{Pa}(x_k^i))$ 
     $\Delta(y_k^i, x_k^i) \leftarrow I(y_k^i = 1) - P(y_k^i = 1 | \mathbf{Pa}(x_k^i))$ 
     $S \leftarrow S \cup [(x_k^i, y_k^i), \Delta(y_k^i, x_k^i)]$ 
  end for
  return  $S$ 
end function

```

2.5 Related work

A number of transfer learning methods in the SRL context have been proposed before. The TAMAR algorithm [14] is one of them. It maps predicates presented in the clauses of an MLN learned from a source domain in order to transfer these clauses to a target domain. The legal mapping that gives the best-weighted pseudo-log-likelihood (WPLL) in the target domain is the mapping used for that clause. In a second step, TAMAR performs theory revision for the mapped structure through an algorithm similar to FORTE algorithm [60] to improve its accuracy.

In order to deal with minimal target data, the algorithm SR2LR [15] was proposed as an extension of TAMAR. It considers the extreme case described as single-entity-centered where one entity is available in the target domain, although they are also generalized for more than one entity. Learning from scratch in the single-entity-centered setting is infeasible, however, transfer learning approach may be very effective if source and target domain are sufficiently related.

Another algorithm, DTM [12], uses second-order Markov Logic where formulas contain predicate variables. The key idea is to discover second-order structure shared by source and target domains by instantiating second-order formulas with predicates from the target domain. It then refines the clauses to better fit the target domain. This refinement may induce new clauses to the theory.

TODTLER algorithm [10] also creates a second-order representation. It uses previous useful second-order patterns learned in the source domain to bias the learning process in the target domain towards models that also have these patterns. TODTLER has shown to be more accurate and considerably faster than DTM. The reason which DTM is slower is basically due to the refinement step present which improves accuracy but it is computationally costly.

Another work is LTL [16] which compares types between source and target predicates and performs a matching. After that, it builds the first-order logic clauses in the target domain by performing a type-based tree construction. LTL algorithm also performs theory refinement in its clauses through the classic theory refinement [89] where the idea is to add or delete predicates in the clauses.

Our algorithm differs from the algorithms presented above mostly on the models used for transference. They rely on MLN models (i.e. set of clauses) for both extracting knowledge from a source model and transferring to a target model. We instead rely on models learned by the algorithm RDN-Boost that uses RRTs for its representation and needs much less training time. We then consider the set of RRTs structured learned from the source domain to bias the learning algorithm in order to obtain a target model derived from the source model. The predicate mapping, necessary for transferring tree structures and inner nodes, is motivated and similar

to TAMAR’s predicate mapping algorithm. Furthermore, a second contribution important to transference is the process of theory revision which is different from the other algorithms since we deal with RRTs instead of purely clauses. Thus, proposing modifications in the theory affects how examples are covered in both the left and the right path. We explain the functioning of the TreeBoostler, our proposed algorithm, in the next chapter.

2.6 Final remarks

In this chapter, we have laid out background knowledge in relational learning, specifically in ILP and SRL. We have also introduced the concept of theory revision and transfer learning which are necessary for the understanding of our proposed algorithm. Then, we introduced the RDN-Boost algorithm and explained its basic concepts. Finally, we pointed out related work in the last section. The next chapter will describe the functioning of the TreeBoostler, our proposed algorithm.

Chapter 3

TreeBoostler: The proposed algorithm

In this chapter, we propose a method that transfers learned boosted trees from a source domain to a target domain. The approach is divided into two major steps: first, the source boosted trees structure is transferred to the target domain by finding an adequate predicate mapping, and second, the algorithm revises its trees by pruning and expanding nodes in order to better fit the target data. The regression values are learned simultaneously in both steps. Next, we detail each of these steps.

3.1 Transferring the structure

A fundamental problem when tackling transfer learning on relational domains is to automatically find how to map the source vocabulary to the target domain. In this way, the first step of the overall process is to find this mapping, where we reduce the overall vocabulary of both domains to their set of predicates, making our first problem as to find the best mapping of source predicates to target predicates. With that, the boosted trees learned from the source domain are transferred sequentially to the target domain and the parameters relearned to fit the target data. MIHALKOVA *et al.* [14] introduced two approaches for establishing a predicate mapping regarding MLNs: (1) a global mapping, which finds a corresponding target predicate to each source predicate and applies this mapping to the entire source structure (i.e. all clauses) at once; and (2) a local mapping, which finds an independent predicate mapping for each independent part of the entire structure (i.e. each clause). This later case constructs a predicate mapping only for the predicates that appear in a specific clause, separately, independently on how the predicates appearing in the other clauses were mapped before. Generally, the local mapping approach is more scalable since the number of predicates that appears in a clause is naturally smaller

than the total number of predicates of a source domain and more flexible, as the mapping in one part of the structure does not necessarily hold or depends on all the other rest of the structure.

In this work, we choose to follow the local approach, by finding the best local predicate mapping for transferring the boosted trees. As we have mentioned earlier, each path from the root to a leaf in the relational regression tree can be considered as a clause in a logic program. However, these paths are not independent of each other as they may share the same inner nodes with different paths in the relational regression tree. In addition, trees cannot be interpreted individually since each one depends on the previously handled trees. Thus, the algorithm translates the predicates presented in the inner nodes according to the previously found translations in order to keep the found predicates mapping through the entire process of learning trees.

3.1.1 Legal mappings

A mapping is legal if each given source predicate is mapped to a compatible target predicate or to an "empty" predicate. If the source and target predicates have the same arity and their argument types agree with the current type constraints they are considered compatible. Mapping is done following the current type constraints which each type mapped to at most one corresponding type in the target domain. For example, the current type constraints are empty and the first predicate to map is $genre(person, genre)$, then the target domain predicate $projectmember(project, person)$ is considered to be compatible. Therefore, the type constraints are updated with the following constraints: $person \rightarrow project$ and $genre \rightarrow person$. Since all sequential predicates to be mapped need to conform to the current type constraints, a mapping for the predicate $advisedby(person, person)$ can only be compatible with $sameproject(project, project)$. A legal mapping is defined in Definition 3.1.1. Algorithm 3 finds legal mappings given the source predicates to be mapped, possible target predicates to consider and current predicate mappings and type constraints.

Definition 3.1.1 *Let $p(X_1, \dots, X_n)$ be an atom in the source vocabulary with predicate p and arity n . Let $q(Z_1, \dots, Z_m)$ be an atom in the target domain with predicate q and arity n . Let $S = \{type_{s_1} \rightarrow type_{t_1} \dots type_{s_n} \rightarrow type_{t_m}\}$ be the set of constrained types, where the first term of each element is a type in the source domain and the second term is a type in the target domain. We say that $p/n \rightarrow q/m$ is a legal mapping when $n = m$ (they have the same arity), and for each pair of corresponding terms (X_i, Z_i) where X_i is a term in $p(X_1, \dots, X_n)$ and Z_i is a term in $q(Z_1, \dots, Z_m)$, if X_i is associated to the type $type_{s_k}$ and Z_i is associated to the*

type $type_{t_j}$, then either $type_{t_j}$ has not appeared before as the second term of an element in S or $type_{s_k} \rightarrow type_{t_j} \in S$. The set of compatible types starts empty and is iteratively filled in with a type correspondence yielded from a predicate mapping.

Algorithm 3 Finding legal mappings given source and target predicates

```

function LEGALMAPPINGS( $srcPreds$ ,  $tarPreds$ ,  $predsMapping$ ,
 $typeConstraints$ )
   $mappingsList \leftarrow []$ 
  Pick the first unmapped source predicate  $srcPred$ 
  for each  $tarPred \in tarPreds$  do
    if  $isCompatible(srcPred, tarPred)$  then
      Add this mapping to a copy of  $predsMapping$ 
      Update a copy of  $typeConstraints$ 
      Call LEGALMAPPINGS with new parameters
      Insert mappings to  $mappingsList$ 
    end if
  end for
  return  $mappingsList$ 
end function

```

Note that the boosted trees are learned with respect to a query atom; because of that, the transfer algorithm must receive as input the source and target query atoms to start the transference. Hence, the predicate mapping starts with a mapping from the source query predicate to the target query predicate. For example, considering to transfer the source query atom $workedunder(person, person)$ from IMDB dataset to the target query atom $advisedby(person, person)$ from UW-CSE dataset, where $person$ is the type of both arguments, in both target query domains. The algorithm starts the type constraints set with the mapping $person \rightarrow person$ and the predicate mapping set with $workedunder(A, B) \rightarrow advisedby(A, B)$. Table 3.1 shows the final predicate mapping set, found after transferring the entire boosted tree structure.

Table 3.1: Found predicate mapping for transferring IMDB→UW-CSE

$workedunder(A, B)$	\rightarrow	$advisedby(A, B)$
$director(A)$	\rightarrow	$professor(A)$
$actor(A)$	\rightarrow	$student(A)$
$movie(A, B)$	\rightarrow	$publication(A, B)$

3.1.2 Finding best mapping and transferring the structure

To find the best predicate mapping for the entire structure, we perform an exhaustive search through the space of all legal mappings of the predicates that are in the inner node which have not been translated yet. The legal mapping that provides to the

node the best split is selected as the best node and mapped predicate. We defined the weighted variance as the split criterion. Transference starts from the root node of the first source tree and proceeds to find not-mapped predicates recursively in order to update the current predicate mapping.

In case the algorithm does not find a compatible mapping, a predicate in the source domain will be mapped to an "empty" predicate. This is used to decide how to map the nodes in the trees, encompassing three cases: (1) all the literals in an inner node have a non-empty predicate mapping. This is the best scenario, as we can keep the same number of literals in the transferred tree; (2) an inner node has some predicate mapped to an "empty" one, but there is at least one predicate mapped to a non-empty, then the ones mapped to empty are discarded and the others remain; (3) an inner node has all their literals mapped to an empty predicate. This is the more complicated scenario, as discarding all the literals yields an empty node, which affects the tree structure, leading to no structure transference in the worst case. For example, the transference $UW-CSE \rightarrow Cora$ would result in a null theory as shown in Figure 3.1 due to the fact that Cora dataset has no unary predicates and the root nodes of learned source trees are conjunctions of unary predicates. To deal with such scenarios the algorithm discards the "empty" node, promotes its left child and appends its right child to the right-most path of the subtree. If the left child is a leaf, then the "empty" node is discarded and the right child is promoted. It is important to mention that the transfer process is also subject to the search bias growing tree parameters, namely the maximum depth and the maximum number of leaves per tree. It means that the nodes and the subtrees appended to the right-most path of the tree may be ignored in the process. In some cases, the transference may result in inner nodes that cover all the examples in their left or right path making the node with no examples useless. To save tree depth, the algorithm discards such nodes and promotes the child that covers all examples. The Algorithm 4 presents the transfer mechanism described.

Our method includes three search bias to conduct the way the algorithm performs the mapping. The first one, called here as *searchArgPermutation*, allows searching for the permutation of all arguments in the target predicate to check if one of them makes the source and target predicates compatible. It allows for example, the mapping of a source predicate with the inverse relation of a target predicate (e.g. $wokedunder(A,B) \rightarrow advises(B,A)$, which is the same as $advisedby(A,B)$). The second search bias, named *searchEmpty*, allows generating an additional "empty" mapping even if there is a compatible target predicate to map the source predicate. The last one, named *allowSameTargetMap*, allows mapping distinct source predicates to the same target predicate. If this bias is not used, the algorithm finds a one-to-one correspondence between source and target predicates (except for "empty")

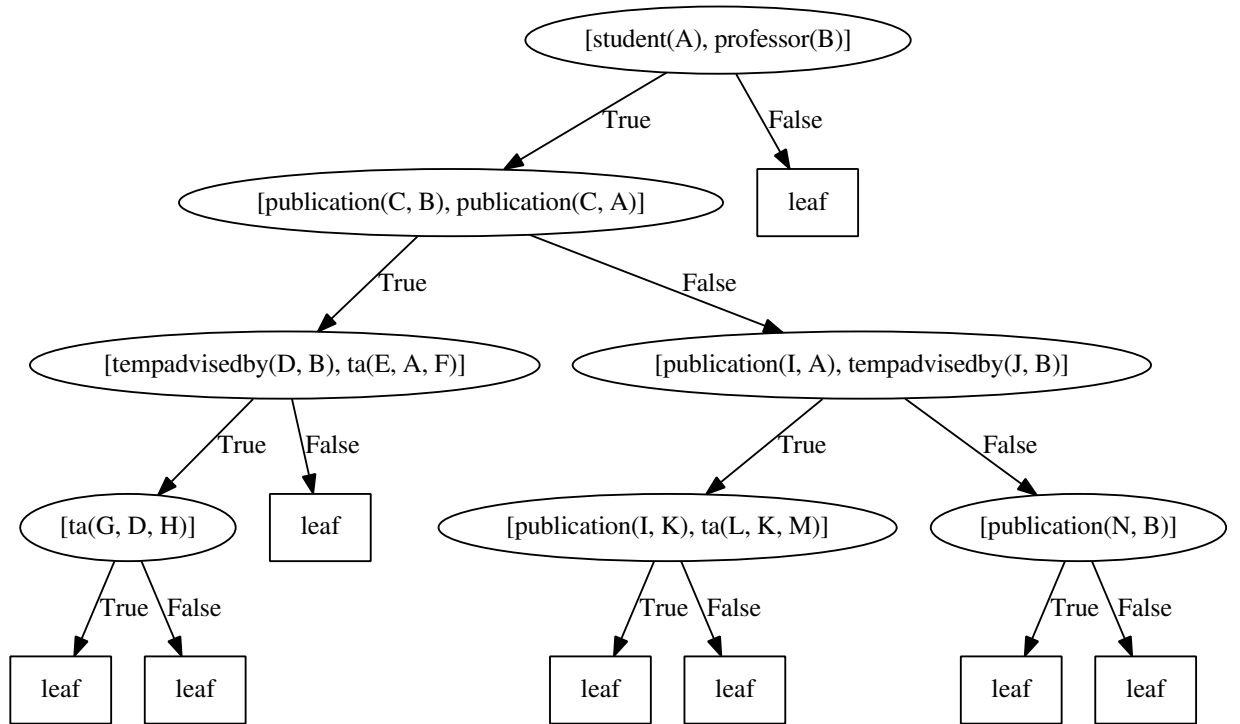


Figure 3.1: One regression tree to be transferred from UW-CSE to Cora for query predicate *advisedby*. Regression values are not considered for transference. They are relearned in the process.

mappings).

3.2 Revising the structure

When transferring learned theories from one domain to another it is usually not enough to map the vocabularies from both domains to achieve a model representative of the target domain [14]. Such theories may contain multiple faults that prevent them to correctly predict examples due to the difference in the distribution of both domains. These faults can be repaired through the process of theory revision [18]. The main idea of theory revision is to search for points in the theory that are preventing the examples to be correctly classified and propose modifications to them. In a transfer learning scenario, the revision process attempts to adjust the initial mapped source theory to fit the target data. The goal is to achieve more accurate theories due to the fact that the theory revision allows the learning algorithm to build clauses from partial or incomplete theories that would otherwise not be found in the constrained search space.

Our theory revision component follows the three major steps:

1. Searching for paths in the trees responsible for bad predictions of examples

Algorithm 4 Top-Level Transfer Algorithm

Require: *theory*, a set of regression trees

Ensure: *transferred*, the transferred regression tree

function TRANSFER(*theory*)

transferred $\leftarrow \emptyset$

for each *tree* \in *theory* **do**

newTree $\leftarrow \emptyset$

 TRANSFERTREE(*tree*, *newTree*)

 Append *newTree* to *transferred*

end for

 return *transferred*

end function

function TRANSFERTREE(*node*, *transferNode*)

if *node* is leaf **then**

 Define *transferNode* as leaf

 Stop procedure

end if

predicates \leftarrow Get set of predicates not mapped from *node*

if *predicates* is empty **then**

newNode \leftarrow Translates predicates in *node*

transferNode \leftarrow *newNode*

else

 Call LEGALMAPPINGS given *predicates* and current predicate mappings and type constraints

 Generate possible nodes by translating predicates in *node* according to legal mappings

 Find the node that gives the best split

 Update the global variable *predsMapping* and *typeConstraints*

transferNode \leftarrow best node

end if

if *transferNode* is not empty **then**

 Call TRANSFERTREE(*node.left*, *transferNode.left*)

 Call TRANSFERTREE(*node.right*, *transferNode.right*)

else

if *node.left* is leaf **then**

 Call TRANSFERTREE(*node.right*, *transferNode*)

else

 Append *node.right* to to the right-most path of *node.left*

 Call TRANSFERTREE(*node.left*, *transferNode*)

end if

end if

end function

and defining them as revision points.

2. Proposing possible modifications to the revision points by applying the revision operators.
3. Scoring both transferred and revised theory and choosing to stay with the best one.

In the traditional theory revision literature concerning ILP, the points to be changed are defined according to a misclassified example defined according to the proved examples, as explained in Section 2.2. However, this concept does not hold for the SRL case which considers the uncertainty of the domain. Thus, we define the points to be changed according to the bad predictions made by the trees. Here, a node is marked as "badly" predicting when its weighted variance is greater than a given threshold δ , reflecting the fact that a node is not good enough to stop the growth of its subtree.

Definition 3.2.1 *Revision Point:* *Let v be a leaf node in a tree and let δ_v be the weighted variance of examples being covered until v . Given a threshold δ , we say that v is "badly" predicting the examples when $\delta_v > \delta$. Hence, the leaf node v is marked as a Revision Point.*

The revision points need to be modified during the revision process in order to increase accuracy. In the traditional ILP setting, examples incorrectly covered determine the revision operator to be applied: a positive example not covered by the theory indicates that the theory is too specific and needs to be generalized, on the other hand, a negative example covered by the theory indicates that the theory is too general and needs to be specialized. In the case of RRTs, positive and negative examples are covered by the paths in the tree, with their respective weights determining the weighted variance of the covered examples. In this way, instead of determining the type of the revision point, as a specialization or a generalization one, we only assume that some paths are responsible for harming the accuracy. To make this matter simpler, we define as a revision point any leaf that has a "bad" weighted variance as defined before. Arguably, modifications on the paths ending up on such leaves will change the way an example is covered resulting in a differently weighted variance.

We considered two types of revision operators: (1) a pruning operator, which increases the coverage of examples by deleting nodes from a tree (and in such a way, it may be seen as a generalization operator); and (2) an expansion operator, which decreases the coverage of examples by expanding nodes in each tree (in the same way, it can be seen as a specialization operator). We describe them as follows:

- **Pruning** operator prunes the tree from the bottom to top by removing a node whose children are leaves marked as revision points
- **Expansion** operator recursively adds nodes that give the best split in a leaf considered as a revision point.

Our Top-Level Theory Revision Algorithm fully applies the pruning and expansion operators in all the revision points at once. The first step is to call the Pruning procedure for each tree in the model. The Pruning procedure receives a root node of a given tree as input and recursively removes nodes that contain leaves marked as revision points. However, this process may completely prune an entire tree eventually leading to the deletion of all the trees particularly because the threshold δ can be very sensitive. If this happens, the revision algorithm would face the expansion of nodes from an empty tree which is the same as learning from scratch. To avoid that, if the pruning results in a null model, the effect of this operator is ignored as if it was never applied.

Next, for each tree, the Expansion procedure is called and recursively expands the revision points. The last step is done by scoring both the transferred theory (before applying theory revision) and the revised theory. The revised theory is implemented in case it has a scoring better than before. The scoring function is the conditional log-likelihood (CLL) over the examples. The Algorithm 5 presents the theory revision process after mapping the vocabulary of the source and target domain.

Next, we provide more details about the revision operators devised in this work.

3.2.1 Pruning

Pruning is a technique that reduces the size of trees by removing nodes of the tree where the bad predictions lie. The pruning operator has two major goals: (1) to cover more examples along a path, which is the equivalent of generalizing clauses, by removing nodes (literals) possibly responsible for bad predictions; and (2) to reduce the size of the trees which may contribute to three additional benefits: (1) improve the inference time, (2) make the trees more interpretable, and (3) help on the rest of the revision process, since it is also subject to tree depth limitations.

The structure of our pruning algorithm is quite simple: it makes a bottom-up pass through a given tree, and decides, for each node, whether to leave the node as it is, or whether to delete this node and make its parent become a leaf. The decision is made considering the successful or failure weighted variance of a path ending in a node. Thus, the algorithm recursively attempts to remove nodes whose children are leaves and revision points, from bottom to up, and keeps subtrees that contain at least one path not marked as a revision point.

Algorithm 5 Top-Level Theory Revision Algorithm

Require: $theory$, a set of regression trees

Ensure: $newTheory$, the possibly revised trees

```
function REVISION( $theory$ )
   $newTheory \leftarrow \emptyset$ 
  for each  $tree \in theory$  do
     $newTree \leftarrow \text{PRUNING}(tree)$ 
    Append  $newTree$  to  $newTheory$ 
  end for
  if  $newTheory$  is null then
     $newTheory \leftarrow theory$ 
  end if
  for each  $tree \in newTheory$  do
     $tree \leftarrow \text{EXPANDNODES}(tree)$ 
  end for
  Compute score  $theory$  and  $newTheory$ 
  if  $score_{newTheory} > score_{theory}$  then
    return  $newTheory$ 
  else
    return  $theory$ 
  end if
end function
```

As mentioned earlier, a node is good enough to stop the growth of its subtree when its weighted variance is less than a given δ . In the opposite way, we consider a node not good enough to remain in the tree when its weighted variance is greater than δ . By removing such a node, we are giving a chance for the algorithm to later find a possible expansion of nodes that would result in better splits. The Pruning operation is presented as Algorithm [6](#).

Algorithm 6 Pruning Operator: Removes nodes recursively if they fit the definition of Revision Point

```
function PRUNING( $node$ )
  left  $\leftarrow \text{PRUNING}(node.left)$ 
  right  $\leftarrow \text{PRUNING}(node.right)$ 
  if left and right child are leaves and both have variance greater than  $\delta$  then
    Remove node from  $node$  and put a leaf in its place
  end if
  return  $node$ 
end function
```

3.2.2 Expansion

The Expansion operator proceeds by adding nodes in an initial theory. As the initial theory is preferably nonempty, as required by the Algorithm [5](#), this process takes

advantage of a starting point, instead of learning from scratch. Adding new nodes and performing splits from starting points may lead to paths that would otherwise not be found in the constrained search space possibly resulting in better covering. Thus, this process is important for two main reasons: (1) by adding nodes in existing paths, it has the same effect of specializing clauses by adding literals to make them more fit to target data; and (2) it takes advantage of the starting point obtained by transference. The expansion is done similarly to the process of learning from scratch; it considers leaves that still need to grow into subtrees as revision points and searches for the node that gives the best split according to the weighted variance as the splitting criterion. The leaves and their regression values are computed when the path is good enough or the tree has reached the maximum depth or number of clauses. Algorithm 7 presents the procedure used here to perform the expansion of nodes.

Algorithm 7 Expansion Operator: Performs expansion of nodes

```

function EXPANDNODES(node)
  left  $\leftarrow$  left child of node
  if left is a leaf and it has variance greater than  $\delta$  then
    Find a new node that gives the best split
    Add this best node to left
    left  $\leftarrow$  EXPANDNODES(left)
  end if
  right  $\leftarrow$  right child of node
  if right is a leaf and it has variance greater than  $\delta$  then
    Find the node that gives the best split
    Add this best node to right
    right  $\leftarrow$  EXPANDNODES(right)
  end if
  return node
end function

```

3.3 Final remarks

In this chapter, we have proposed a transfer learning approach that transfers learned boosted trees from a source domain to a target domain. We have pointed out the contributions of this approach which includes a transfer system that maps source predicates to target predicates and a revision theory system that proposes modifications to the boosted trees in order to better fit the target data. In the next chapter, we will introduce the experiments conducted in order to analyze our algorithm against baseline approaches.

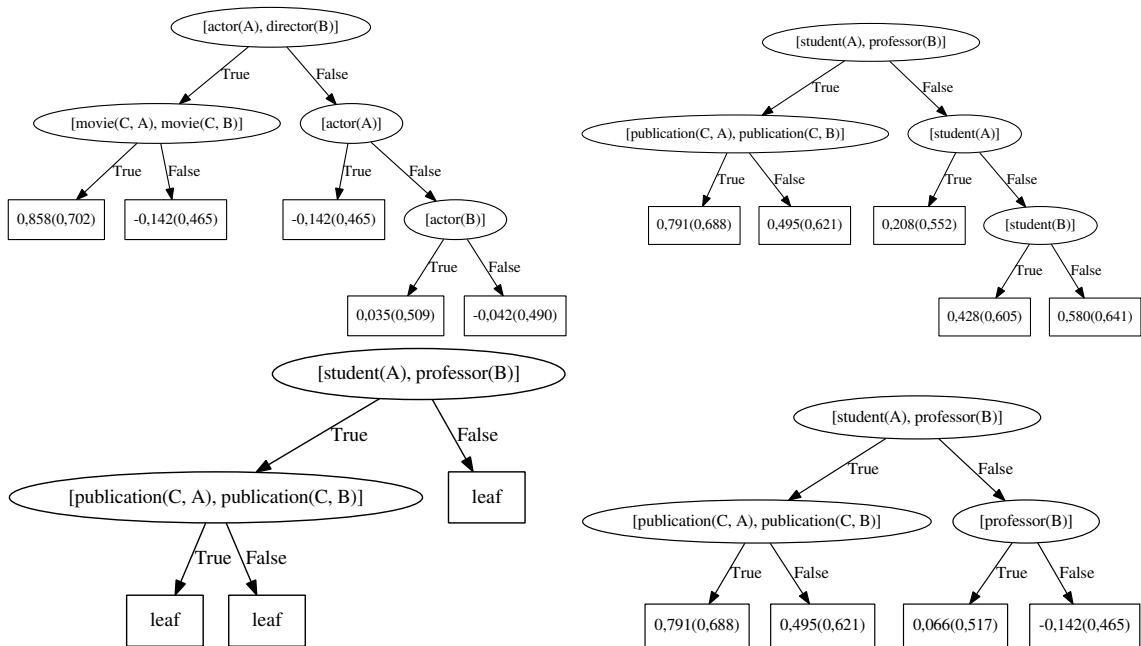


Figure 3.2: The transfer learning process stages. The trees presented are the following: obtained from source domain by learning from scratch (top-left); transferred by mapping predicates (top-right); after the pruning process (down-left) and after the expansion of nodes (down-right). All trees are the first one learned in the iterations. The transference is done from IMDB to UW-CSE and depth limits were reduced to generate smaller trees. Regression values are not considered in the pruning process and they are relearned when expanding nodes.

Chapter 4

Experiments and results

In this chapter, we present the experiments we conducted in this work in order to investigate the research questions presented in this chapter. We have performed three types of experiments: (1) an approach simulating a transfer learning environment with limited target data, (2) a scenario with increasing amounts of target data and (3) a scenario that represents learning from minimal target data.

4.1 Research questions

We conducted the experiments in order to investigate the following research questions considering both transfer learning and learning from scratch baselines:

- **Q1:** Does TreeBoostler learn more accurate models than the baselines?
- **Q2:** Does theory revision improve the performance of the transfer process?
- **Q3:** Does TreeBoostler transfer well across domains?
- **Q4:** Is TreeBoostler faster than the baselines?
- **Q5:** Does TreeBoostler perform better than the baselines with increasing amount of examples in the target data?
- **Q6:** Does TreeBoostler perform better than the baselines with minimal target data?

The question **Q1** addresses a common question when comparing different algorithms. It is important to evaluate the algorithm and conclude if it performs better than learning from scratch approaches and related transfer learning approaches. Question **Q2** is important to evaluate the effectiveness of a theory revision process and demonstrates that the process is capable of improving the performance of the transferred model in the target domain. This research question was also addressed

in [16]. The question Q3 addresses if the transfer process is capable of providing good models while question Q4 asks if the algorithm is faster than related transfer learning algorithms and learning from scratch algorithms. It would be desirable to provide a transfer learning approach that could be faster than learning from scratch. Since the transfer learning system is provided with a trained model from the source domain, part of the time-consuming in the learning process is already done. The question Q5 addresses how the algorithms behave with increasing amounts of data. This question was also addressed in [10, 12, 14, 16] through learning curves describing the accuracy in different numbers of mega-examples. The question Q6 addresses the problem of minimal target data where the learner is provided with only a few examples. This problem was the motivation of SR2LR [15] which addressed the *single-entity-centered* setting in which the learner is provided with information concerning only a single entity, i.e. background knowledge contains only information related to the single entity. Differently, in this work, we provided to the learner all the background knowledge available but only a few positive and negative examples.

4.2 Datasets

Following previous literature, we present our results considering six different publicly available datasets described as follows.

- The Cora dataset [90] is a database of Computer Science citations with 1295 different citations to 112 computer science research papers and has fields as author, venue, title and year. The goal in this dataset is to predict the *samevenue* relation which determines if two venues represent the same conference. This dataset is divided into 5 mega-examples.
- The UW-CSE dataset [91] consists of information about professors, students, and courses from 5 different areas of computer science (artificial intelligence, programming languages, theory, system, and graphics). Thus, this dataset is divided into 5 mega-examples. It includes predicates that represent publications and their authors, projects and their members, level of courses, etc. The goal is to predict the *advisedby* relation which identifies a student being advised by a professor.
- IMDB [92] is a dataset that describes a movie domain and presents predicates as director, actor, genre, movie, etc and the relationships between them. It is divided into 5 mega-examples where each one contains information about 4 movies. The goal is to predict the *workedunder* relation which identifies an actor that has worked for a director.

- The Yeast protein [9] dataset is obtained from MIPS [1] Comprehensive Yeast Genome Database and includes information about proteins with their location, function, phenotype, enzyme, etc. The goal in this dataset is to predict the class of a protein. This dataset contains 4 folds independent of each other.
- Twitter [10] is a dataset that contains tweets about Belgian soccer matches. The information is basically words that are tweeted, relations between accounts (following relation) and the type of accounts (club, fan or news). The goal is to predict the type of an account in 2 independent folds.
- NELL [93] is a machine learning system that extracts probabilistic knowledge base from online text data. We consider only two domains from NELL dataset, which are the Sports domain, extracted from the iteration 1070 and the Finances domain, extracted from the iteration 1115. The goal in Sports domain is to predict the relation that defines a team playing a sport and in the Finances domain, the goal is to predict the relation that defines a company belonging to an economic sector. In order to obtain different folds, we split the data of the target predicate randomly into 3 parts. Thus, each fold consists of parts of the target predicates and all facts (non-target predicates).

Table 4.1 provides additional statistics about these datasets. The number of true ground literals is the number of true examples of the respective target predicate of each dataset. The total number is the number of all ground literals formed by grounding the predicates with constants of their respective types.

Table 4.1: Statistics about datasets

Dataset	Number of Constants	Number of Types	Number of Predicates	Number of true ground literals	Total number of ground literals
Cora	2457	5	10	3017	152100
UW-CSE	919	9	14	113	16900
IMDB	297	3	6	382	71824
Yeast	2470	7	7	369	40128
Twitter	273	3	3	282	663
NELL Sports	4538	4	8	397	4323
NELL Finances	3340	5	10	778	51578

4.3 Methodology and results

We compared the performance of TreeBoostler against two baseline approaches that learn from scratch from target data: RDN-B, which learns a set of regression trees

¹Munich Information Center of Protein Sequence

using boosting method and RDN which learns a single large regression tree. In this chapter, we will refer to RDN-Boost as RDN-B. We also compared it against TODTLER [10], a transfer learning method that lifts a source structure to second-order logic. We did not compare against the state-of-the-art algorithm LTL [16] because only the system responsible for transferring the clauses was available. The system for parameter learning using weighted-mean as the combination function of rules and the system for performing local theory refinement are crucial to produce competitive results for the algorithm LTL.

To observe if the revision stage improves the performance of the whole transference process, two stages of the algorithm are considered: transference considering predicates mapping and parameter learning only, i.e. the first stage of the complete algorithm (TreeBoostler*) and the complete transfer system using predicate mapping and theory revision (pruning and expansions of trees) (TreeBoostler). For TreeBoostler, we restricted the depth limit of the trees to be 3, the number of leaves to be 8, the number of regression trees was 10, and the maximum number of literals per node was restricted to 2. We used the same settings to learn from scratch using the method RDN-B. For the single tree RDN method, we used 20 leaves. For the threshold used in the theory revision step, we set its value as 2.5×10^{-3} which is the same value used as default in the RDN and RDN-B algorithms to decide if a variance of a node is small enough to become a leaf in the learning process. For training all the RDN based algorithms, we subsampled the negatives examples in a ratio of two negatives for one positive. Thus, following [17], we set the initial potential to be -1.8. For testing, we presented all the negative examples. For the MLN based approach TODTLER, we used Alchemy with default settings and MC-SAT algorithm (option *-ms*) to compute the probabilities. Also, we kept the default parameters and generated second-order templates containing at most three literals and three object variables.

For all our experiments, we allowed TreeBoostler to search for all permutations of arguments of a given predicate. This was very important for transferring among NELL datasets since some source predicates can be considered as the inverse of a possible mapped target predicate. Also, we did not allow more than one distinct source predicate to be mapped to the same target predicate, as this bias does not improve the results while still increases the training time. The option *searchEmpty* was also set to false to avoid increasing the amount of training time.

The first experiment was done in order to simulate the learning process from very limited data which is the more suitable scenario to transfer learning. We employed the same methodology used in related works [10, 12, 16]: training is performed on 1 fold and testing on the remaining $n - 1$ folds. The results are then averaged over n runs. For each run, a new learned source model is used for transference. Specifically

for TODTLER, the results were obtained from one single run due to extremely time-consuming resources when computing scores for each first-order clause using Alchemy. TODTLER was not able to finish computing scores for clauses in NELL dataset after one week. We used the following measures to compare the performance: conditional log-likelihood (CLL), the area under the ROC curve (AUC ROC), the area under the PR curve (AUC PR) and training time. Note that in the training time of transfer systems we did not consider the time necessary to learn from the source domain.

The results are presented in the Tables [4.2](#), [4.3](#) and [4.4](#). The Tables [4.2](#), [4.3](#) present the transfer experiment for the pairs of datasets IMDB and Cora and also Yeast and Twitter, respectively. Each dataset was treated as a source domain and target domain in the experiments. The Table [4.4](#) presents the transfer experiments IMDB \rightarrow UW-CSE and NELL Sports \rightarrow NELL Finances. We omitted the opposite transfer experiments because transferring from UW-CSE to IMDB is too easy, including learning from scratch from IMDB, and the transference from NELL Finances to NELL Sports which resulted in negative transfer because it was unable to find useful mappings. It can be observed that our algorithms are competitive or better than TODTLER and learning from scratch methods. Our algorithms and learning from scratch methods outperform TODTLER in most of the results presented, mostly due to the efficiency and expressiveness of the language used for representing RDNs. Therefore, it is more interesting to compare our algorithms against learning from scratch methods. The TreeBoostler algorithm performed comparably or better than learning from scratch methods in all but one experiment for AUC ROC. Even for the TreeBoostler*, which is restricted only for mapping, was able to learn more accurate models than learning from scratch in 2 experiments for AUC ROC and 3 for AUC PR. Then only mapping the predicates and learning the parameters for the mapped trees may be very useful when target training data is scarce. The most significant result can be observed in the transference from the real-world dataset NELL Sports to NELL Finances. Bold results are significantly better than the performance of all baselines (RDN, RDN-B and TODTLER) for at least one TreeBoostler algorithm. The statistical significance was measured using a paired t-test at the 95% confidence level. Based on these experiments and observations, we can positively answer the questions **Q1** and **Q3** posed before.

As can be seen, the training time consumed by TreeBoostler* is usually smaller than RDN-B and equivalent to RDN. This is because the transfer algorithm only needs to find the best split for those nodes that have not-mapped predicates, otherwise it already knows which mapped node to consider in the split, avoiding searching and evaluating other possible mappings. The first time a predicate appears in the set of regression trees is the only time a mapping has to be found for this predi-

Table 4.2: Results on IMDB and Cora dataset. We compare our algorithm, RDN-B (that uses boosting), RDN and TODTLER. We present the results for the area under curves for ROC and PR and the conditional log-likelihood for test examples. We also present the training time.

Algorithm	IMDB \rightarrow Cora				Cora \rightarrow IMDB			
	CLL	AUC ROC	AUC PR	Time	CLL	AUC ROC	AUC PR	Time
RDN	-0.192	0.641	0.074	16.70 s	-0.166	0.994	0.813	1.17 s
RDN-B	-0.277	0.842	0.270	237.47 s	-0.073	1.000	1.000	3.29 s
TODTLER	-5.213	0.519	0.371	17 min	-0.923	0.885	0.537	195.77 s
TreeBoostler*	-0.323	0.582	0.183	5.34 s	-0.213	0.958	0.727	2.82 s
TreeBoostler	-0.298	0.707	0.292	106.39 s	-0.077	0.999	0.952	10.70 s

Table 4.3: Results on Yeast and Twitter dataset. We present the results for the area under curves for ROC and PR, the conditional log-likelihood and the training time.

Algorithm	Yeast \rightarrow Twitter				Twitter \rightarrow Yeast			
	CLL	AUC ROC	AUC PR	Time	CLL	AUC ROC	AUC PR	Time
RDN	-0.155	0.964	0.271	4.08 s	-0.182	0.695	0.081	4.46 s
RDN-B	-0.118	0.993	0.382	24.42 s	-0.257	0.919	0.231	18.80 s
TODTLER	-1.259	0.520	0.368	13.42 s	-0.023	0.497	0.002	39 min
TreeBoostler*	-0.138	0.986	0.394	6.12 s	-0.180	0.986	0.273	4.14 s
TreeBoostler	-0.118	0.993	0.362	114.71 s	-0.180	0.986	0.272	60.99 s

Table 4.4: Results on transference from IMDB to UW-CSE dataset and NELL Sports domain to Finances domain considering area under the curves for ROC and PR, the conditional log-likelihood and the training time.

Algorithm	IMDB \rightarrow UW-CSE				NELL Sports \rightarrow NELL Finances			
	CLL	AUC ROC	AUC PR	Time	CLL	AUC ROC	AUC PR	Time
RDN	-0.194	0.918	0.247	1.79 s	-0.180	0.532	0.020	4.59 s
RDN-B	-0.261	0.935	0.265	8.17 s	-0.317	0.713	0.083	22.12 s
TODTLER	-3.699	0.570	0.037	208 min	NA	NA	NA	NA
TreeBoostler*	-0.274	0.926	0.275	1.16 s	-0.164	0.978	0.062	46.63 s
TreeBoostler	-0.241	0.940	0.305	9.20 s	-0.161	0.979	0.074	229.36 s

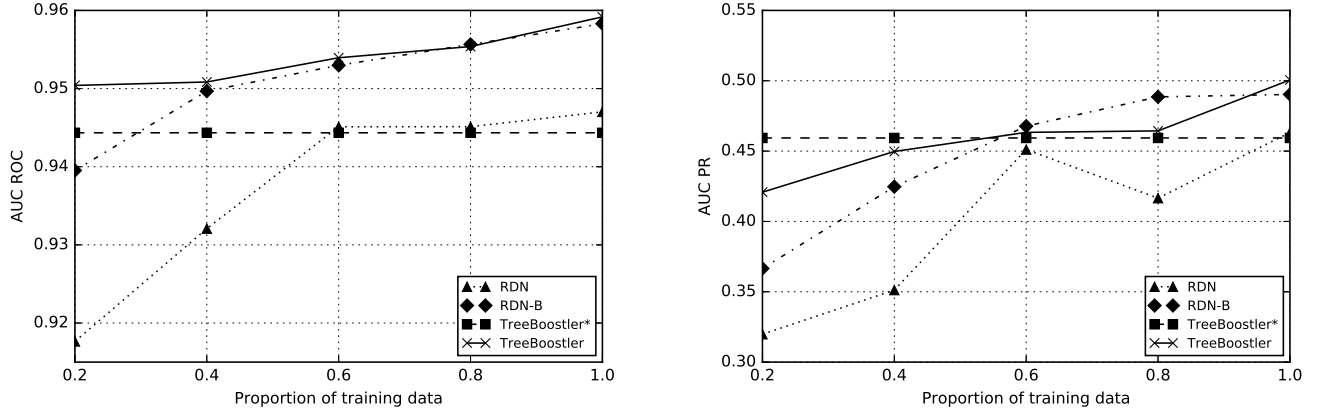


Figure 4.1: Learning curves for AUC ROC (left) and AUC PR (right) obtained from IMDB \rightarrow UW-CSE.

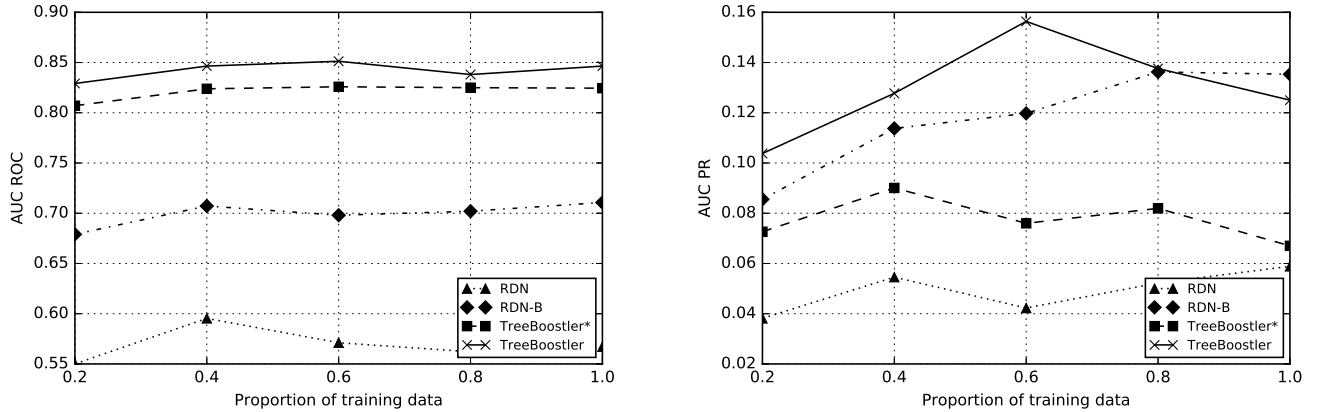


Figure 4.2: Learning curves for AUC ROC (left) and AUC PR (right) obtained from NELL Sports \rightarrow NELL Finances.

cate. It saves time in the rest of the tree and the next iterations as the algorithm knows how to transfer a source inner node. On the other hand, TreeBoostler, considering theory revision, improves accuracy but is computationally costly since it is another search approach. This training time considers the time spent in the entire process which includes the time taken for transference, the time taken for evaluating both the transferred and the revised model, and the time taken for pruning and expansion. In summary, we can answer **Q4** affirmatively for TreeBoostler* and affirmatively comparing to other transfer learning system for TreeBoostler. The results show that question **Q2** can also be answered positively. The theory revision process shows an improvement in the performance for all the metrics except for a worse AUC PR in a single experiment.

In order to compare the performance of our method with increasing amounts of

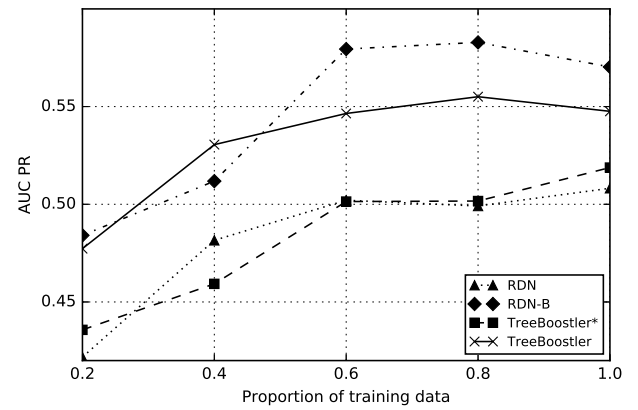
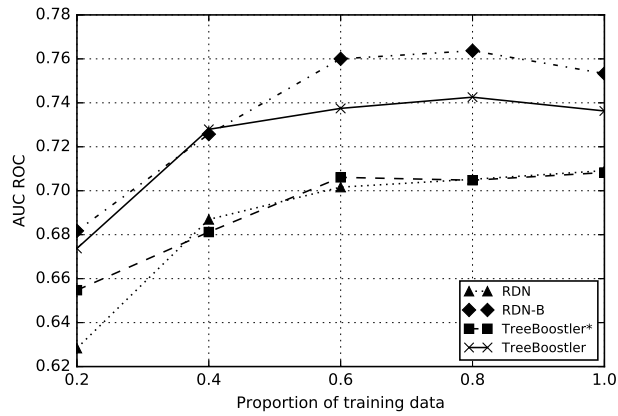


Figure 4.3: Learning curves for AUC ROC (left) and AUC PR (right) obtained from Yeast \rightarrow Twitter.

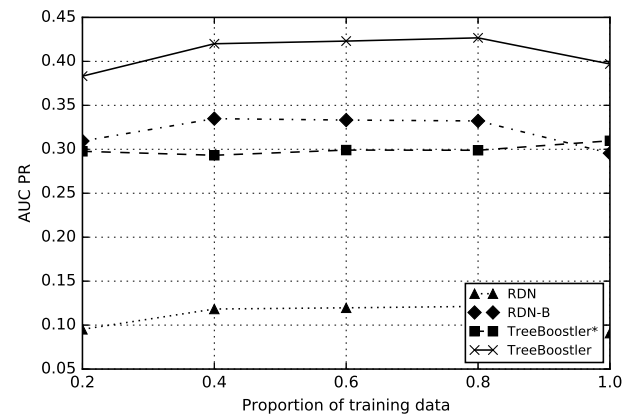
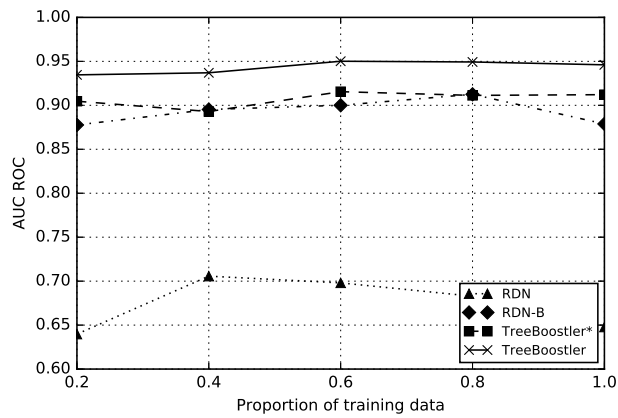


Figure 4.4: Learning curves for AUC ROC (left) and AUC PR (right) obtained from Twitter \rightarrow Yeast.

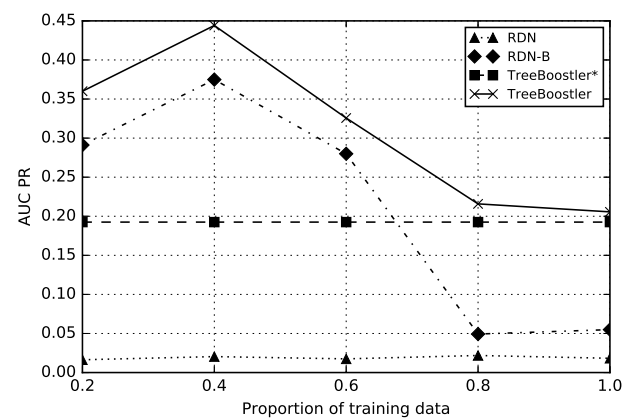
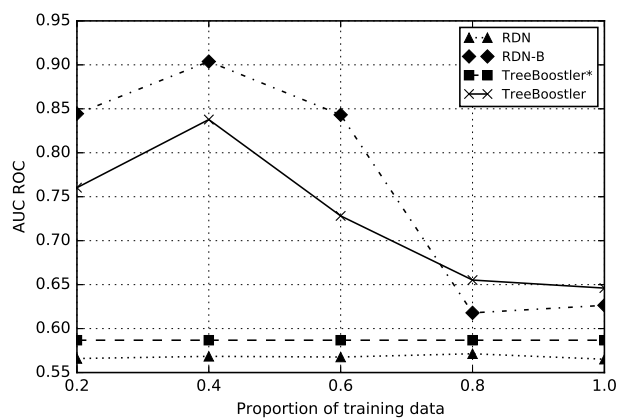


Figure 4.5: Learning curves for AUC ROC (left) and AUC PR (right) obtained from IMDB \rightarrow Cora.

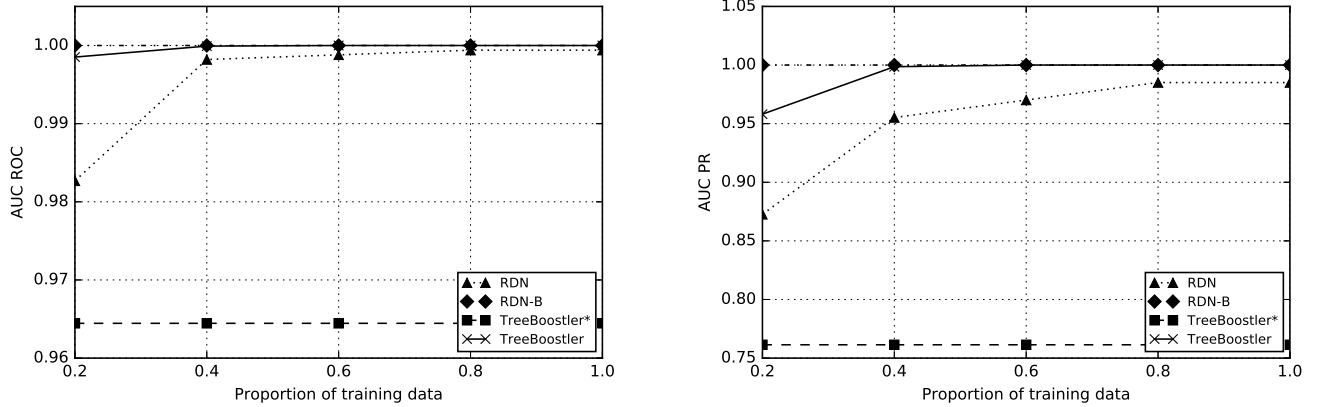


Figure 4.6: Learning curves for AUC ROC (left) and AUC PR (right) obtained from Cora \rightarrow IMDB.

target data, we performed a learning curve experiment transferring the same pairs of datasets. For these experiments, we employed the traditional cross-validation methodology when training is performed on $n-1$ folds and testing on the remaining 1 fold. The data selected for training is then shuffled and divided into 5 sequence parts. All systems observed the same sequence of these parts. The entire process is done in n runs and the curves are obtained by averaging the results. The Figures [4.1](#), [4.2](#), [4.3](#), [4.4](#), [4.5](#) and [4.6](#) demonstrate this experiment. As can be seen, our algorithm outperforms or equates learning from scratch RDN-B in most of the results, particularly with smaller amounts of data (about 40% of the target data). An exception is the learning curve for the AUC ROC in Figure [4.6](#) which demonstrates a decreasing in the performance as the target data increases. In this experiment, TreeBoostler is outperformed by RDN-B until 80% of the target data, although it outperforms RND-B in terms of AUC PR. Thus, question **Q5** can be answered affirmatively.

A third experiment was conducted in order to address the problem of minimal target data and investigate how the algorithms behave when learning from only a few examples. We also performed a learning curve experiment with the same pairs of datasets. We employed the traditional cross-validation methodology, then we shuffled the data for training and selected 5 groups of 5 positive examples and 5 groups of 5 negative examples. All systems observed the same sequence of these groups of examples, i.e. systems observed from 5 up to 25 examples for each label. Similarly to the last experiment, the entire process is done in n runs and the curves are obtained by averaging the results. The Figures [4.7](#), [4.8](#), [4.9](#), [4.10](#), [4.11](#) and [4.12](#) demonstrate this experiment. As indicated in the experiments, our algorithms easily outperform the learning from scratch algorithms RDN-B and RDN in all the presented results. The small amount of training data available was insufficient to

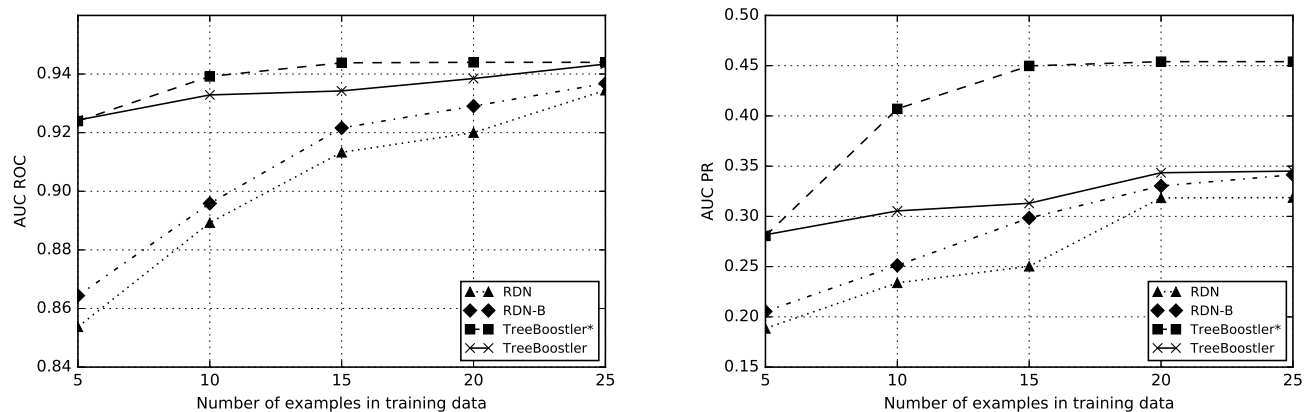


Figure 4.7: Learning curves from minimal target data for AUC ROC (left) and AUC PR (right) obtained from IMDB \rightarrow UW-CSE.

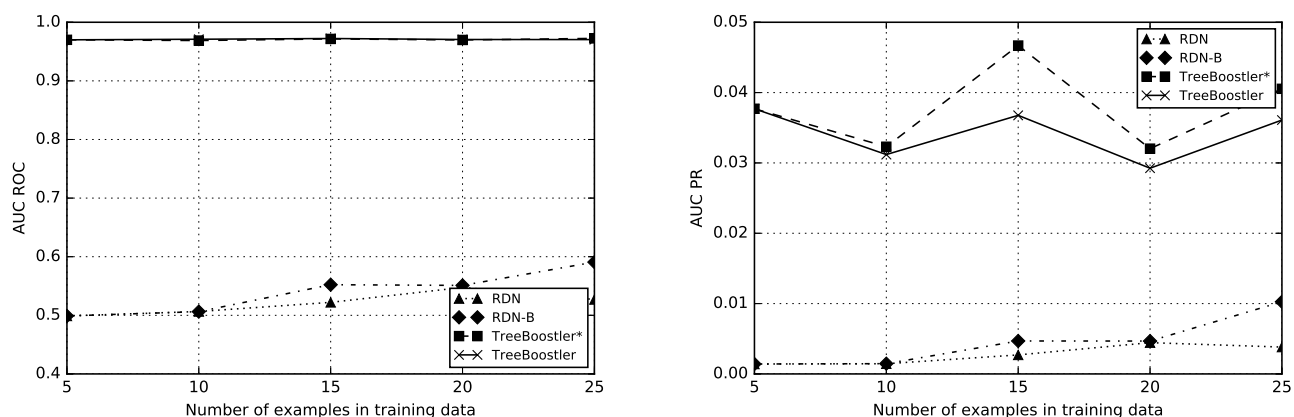


Figure 4.8: Learning curves from minimal target data for AUC ROC (left) and AUC PR (right) obtained from NELL Sports \rightarrow NELL Finances.

learn good models in learning from scratch approaches. Providing more examples has shown to increase the performance of these approaches, however it was still insufficient comparing to TreeBoostler which also increased its performance when more examples are provided. As can be seen, the revision step also showed to slightly decrease the performance in the experiments, except for the experiment in Figure [4.11](#) and [4.12](#). This may be basically due to difficulty of revising and simultaneously relearning parameters of models given very few examples. Since the pruning and expansion operators are subject to the threshold δ , very few examples may not be sufficient to determine correctly when a node is "badly" predicting. Thus, according to these experiments, we can answer question **Q6** positively.

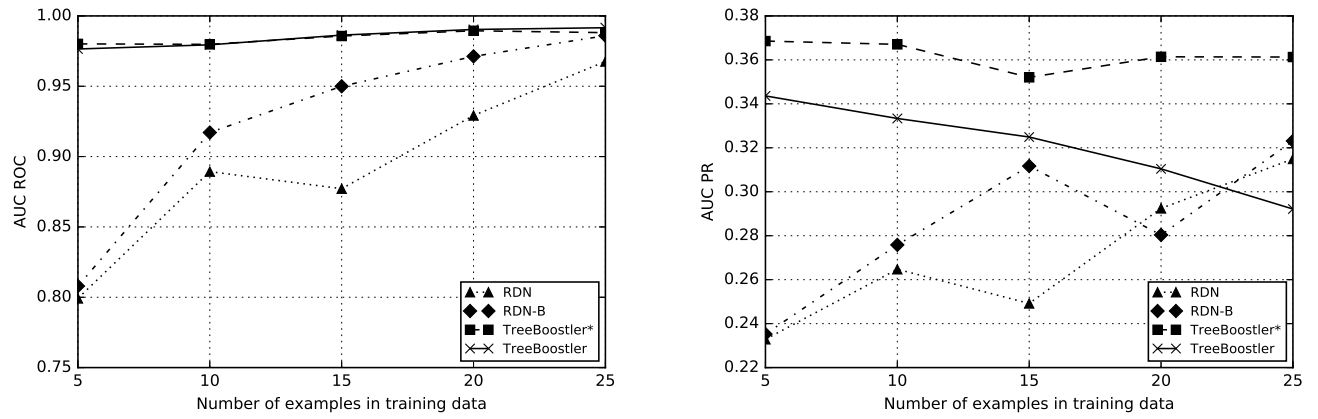


Figure 4.9: Learning curves from minimal target data for AUC ROC (left) and AUC PR (right) obtained from Yeast \rightarrow Twitter.

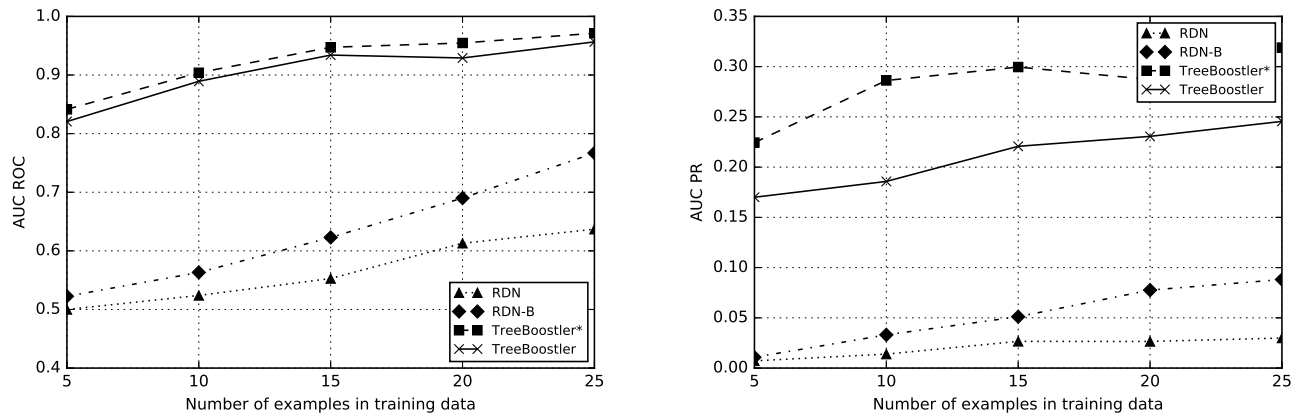


Figure 4.10: Learning curves from minimal target data for AUC ROC (left) and AUC PR (right) obtained from Twitter \rightarrow Yeast.

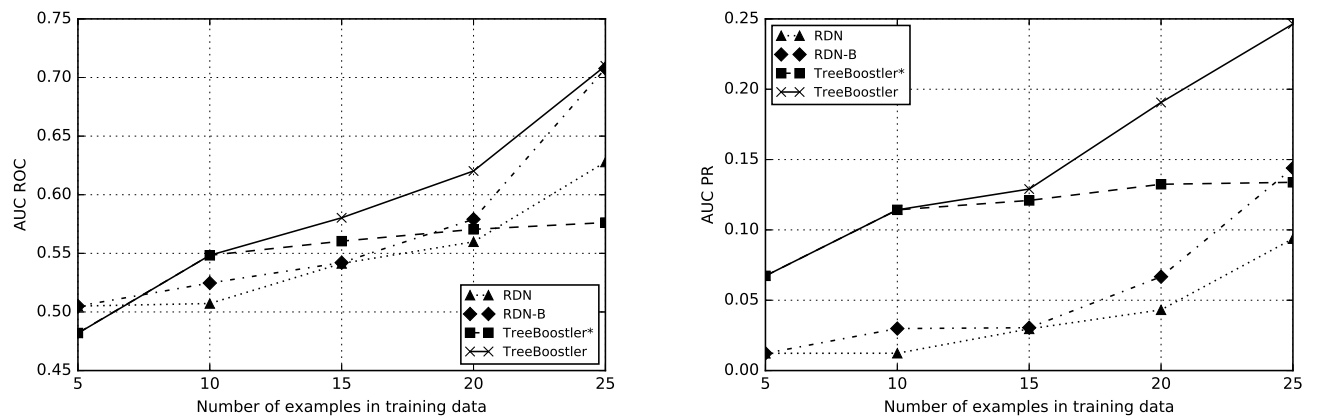


Figure 4.11: Learning curves from minimal target data for AUC ROC (left) and AUC PR (right) obtained from IMDB \rightarrow Cora.

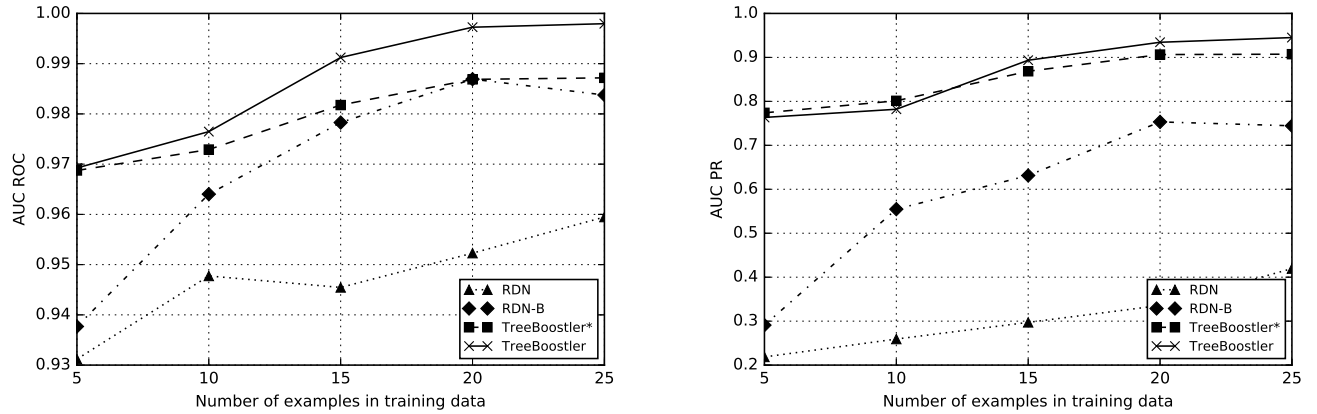


Figure 4.12: Learning curves from minimal target data for AUC ROC (left) and AUC PR (right) obtained from Cora \rightarrow IMDB.

4.4 Final remarks

In this chapter, we have experimented and analyzed the proposed algorithm against learning from scratch approaches and one related transfer learning algorithm TODTLER. We have evaluated the algorithms simulating the learning process from very limited data employing the same methodology used in related works. We also performed experiments on a scenario with increasing amounts of target data and a scenario that represents learning from minimal target data. Our TreeBoostler algorithm outperformed the baselines introduced in this section.

In the following and final chapter, we will look back on the methods proposed in this work and provide possible future research directions.

Chapter 5

Conclusion

In this work, we have proposed a novel transfer learning method, named as TreeBoostler, that transfers Boosted RDNs learned from a source domain to a desirable target domain. TreeBoostler constructs a target set of regression trees biased by a predicate mapping found through the transfer process given the structure of the source regression trees. Then, it applies a second stage process relying on theory revision, to propose modifications to the mapped model. These modifications are done through two proposed revision operators for the regression trees which are the pruning operator and the expansion operator. The pruning operator showed to be important for deleting nodes in the tree and providing space for the expansion of new nodes.

Through experimental results, we found out that even the first state of the entire transfer process, which only maps predicates and learn the parameters of them, can give better results than learning from scratch in a smaller amount of training time. The application of theory revision in transfer learning approaches brings the benefit of handling incorrectness that has come from a different yet related domain through transference. Thus, a transference may not provide a good model due to differences between domains. However, this incorrectness may be successfully handled by the modifications proposed in the system.

Our experimental results demonstrate that this algorithm is effective compared to the other transfer algorithm TODTLER mainly because of the efficiency and expressiveness of the language used for representing RDNs. We have performed the experiments by simulating a transfer learning scenario where only a few data are available. We showed from experiments that transfer learning may result in much more accurate models compared to learning from scratch methods, however, it may also hurt the learning performance and result in less accurate models. Moreover, the theory revision process improved the performance of the transferred models showing the effectiveness of proposing modifications to fit the model to the target data. However, the experiments also showed that theory revision is more time

consuming since it is another search approach. According to the experiments, our algorithm demonstrated to be as much efficient as learning from scratch methods.

5.1 Future work

A possible future direction is to take advantage of stochastic search methods [62] in the process of pruning, allowing the method to generate different pruning in the trees at random and expand nodes from these candidates. This may help the algorithm to escape from a local optimum since the pruning process follows a deterministic criteria. Stochastic search methods can also benefit the process of expanding nodes by allowing different possible expansions in the path instead of finding the best node for split.

Another possible research direction is developing the transfer learning process to regression trees that contain predicates with constants or numeric values, increasing the expressiveness of the language used for transference. Our algorithm and related work does not allow transfer when clauses contain either constants or numeric values. This can be very beneficial since constants from different domains may also be related, e.g. a particular movie genre may represent well a particular field of study. Also, numeric values from different domains may represent different but related measures.

In addition, the investigation and development of new revision operators for boosted trees could lead to a possible research direction. New revision operators may lead to improvements in performance of a model since it would increase the number of modifications proposed. However, differently from revising separated clauses as many works have proposed, the paths in the trees are not independent of each other and they may share a same inner node. The trees are not independents as well, since each one depends on the previously handled trees. Then, the research direction has to consider this particularity of revising boosted trees.

Finally, it is also interesting to investigate how to compute the similarity between domains beforehand to avoid negative transfer. The development of a measure may help to avoid struggling when transferring knowledge between domains that may be related or not.

Bibliography

- [1] LEE, K., CAVERLEE, J., WEBB, S. “Uncovering social spammers: social honeypots+ machine learning”. In: *Proceedings of the 33rd international ACM SIGIR conference on Research and development in information retrieval*, pp. 435–442. ACM, 2010.
- [2] SINAPOV, J., STOYTCHEV, A. “Object category recognition by a humanoid robot using behavior-grounded relational learning”. In: *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pp. 184–190. IEEE, 2011.
- [3] RUSU, A. A., VECERIK, M., ROTHÖRL, T., et al. “Sim-to-Real Robot Learning from Pixels with Progressive Nets”, *CoRR*, v. abs/1610.04286, 2016.
- [4] PAN, J., ZHENG, W., YANG, Q., et al. “Transfer learning for WiFi-based indoor localization”. In: *AAAI 2008*, 2008.
- [5] BLITZER, J., DREDZE, M., PEREIRA, F. “Biographies, bollywood, boomboxes and blenders: Domain adaptation for sentiment classification”. In: *In ACL*, pp. 187–205, 2007.
- [6] PAN, S. J., YANG, Q. “A Survey on Transfer Learning”, *IEEE Trans. on Knowl. and Data Eng.*, v. 22, n. 10, pp. 1345–1359, 2010.
- [7] DAI, W., YANG, Q., XUE, G.-R., et al. “Boosting for Transfer Learning”. In: *Proceedings of the 24th International Conference on Machine Learning, ICML '07*, pp. 193–200, New York, NY, USA, 2007. ACM. ISBN: 978-1-59593-793-3.
- [8] W. LEE, J., GIRAUD-CARRIER, C. “Transfer Learning in Decision Trees”. In: *2007 International Joint Conference on Neural Networks*, pp. 726–731, Aug 2007.
- [9] MEWES, H. W., HEUMANN, K., KAPS, A., et al. “MIPS: a database for genomes and protein sequences”, *Nucleic Acids Research*, v. 27, n. 1, pp. 44–48, 1999.

- [10] VAN HAAREN, J., KOLOBOV, A., DAVIS, J. “TODTLER: Two-order-deep transfer learning”. In: *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, AAAI Conference on Artificial Intelligence*, 2015.
- [11] GETOOR, L., TASKAR, B. *Introduction to Statistical Relational Learning (Adaptive Computation and Machine Learning)*. The MIT Press, 2007. ISBN: 0262072882.
- [12] DAVIS, J., DOMINGOS, P. “Deep Transfer via Second-Order Markov Logic”. In: *Proceedings of the 26th International Conference on Machine Learning (ICML-09)*, 2009.
- [13] RICHARDSON, M., DOMINGOS, P. “Markov Logic Networks”, *Mach. Learn.*, v. 62, n. 1-2, pp. 107–136, fev. 2006. ISSN: 0885-6125.
- [14] MIHALKOVA, L., HUYNH, T., MOONEY, R. J. “Mapping and Revising Markov Logic Networks for Transfer Learning”. In: *Proceedings of the 22Nd National Conference on Artificial Intelligence - Volume 1, AAAI’07*, pp. 608–614. AAAI Press, 2007.
- [15] MIHALKOVA, L., MOONEY, R. “Transfer Learning from Minimal Target Data by Mapping across Relational Domains”. In: *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI-09)*, pp. 1163–1168, Pasadena, CA, July 2009.
- [16] KUMARASWAMY, R., ODOM, P., KERSTING, K., et al. “Transfer Learning via Relational Type Matching”. In: *Proceedings of the 2015 IEEE International Conference on Data Mining (ICDM)*, ICDM ’15, pp. 811–816. IEEE Computer Society, 2015. ISBN: 978-1-4673-9504-5.
- [17] NATARAJAN, S., KHOT, T., KERSTING, K., et al. “Gradient-based Boosting for Statistical Relational Learning: The Relational Dependency Network Case”, *Mach. Learn.*, v. 86, n. 1, pp. 25–56, jan. 2012. ISSN: 0885-6125.
- [18] WROBEL, S. “First Order Theory Refinement”. In: *Advances in inductive logic programming*, IOS Press, 1996.
- [19] LAVRAC, N., DZEROSKI, S. *Inductive Logic Programming: Techniques and Applications*. New York, NY, 10001, Routledge, 1993. ISBN: 0134578708.
- [20] MUGGLETON, S. “Inductive Logic Programming”, *New Gen. Comput.*, v. 8, n. 4, pp. 295–318, fev. 1991. ISSN: 0288-3635.

- [21] BRATKO, I. *Prolog Programming for Artificial Intelligence*. 3 ed. Harlow, England, Pearson Addison-Wesley, 2000. ISBN: 978-0-201-40375-6.
- [22] LLOYD, J. W. *Foundations of Logic Programming*. Berlin, Heidelberg, Springer-Verlag, 1984. ISBN: 0-387-13299-6.
- [23] NIENHUYS-CHENG, S.-H., WOLF, R. D. *Foundations of Inductive Logic Programming*. Berlin, Heidelberg, Springer-Verlag, 1997. ISBN: 3540629270.
- [24] DOLSAK, B., MUGGLETON, S. “The Application of Inductive Logic Programming to Finite Element Mesh Design”. In: *Inductive Logic Programming*, pp. 453–472. Academic Press, 1992.
- [25] FENG, C. “Inducing Temporal Fault Diagnostic Rules from a Qualitative Model”. In: *Proceedings of the Eighth International Conference on Machine Learning*, ML’91, pp. 403–406, San Francisco, CA, USA, 1991. Morgan Kaufmann Publishers Inc. ISBN: 1-55860-200-3.
- [26] HAU, D. T., COIERA, E. W. “Learning Qualitative Models of Dynamic Systems”, *Machine Learning*, v. 26, n. 2, pp. 177–211, Feb 1997. ISSN: 1573-0565.
- [27] KING, R., MUGGLETON, S., LEWIS, R., et al. “Drug Design by Machine Learning: The Use of Inductive Logic Programming to Model the Structure-Activity Relationships of Trimethoprim Analogues Binding to Dihydrofolate Reductase”, *Proceedings of the National Academy of Sciences of the United States of America*, v. 89, pp. 11322–6, 01 1993.
- [28] MUGGLETON, S., KING, R. D., STENBERG, M. J. “Protein secondary structure prediction using logic-based machine learning”, *Protein Engineering, Design and Selection*, v. 5, n. 7, pp. 647–657, 10 1992. ISSN: 1741-0126.
- [29] LAVRAC, N., DZEROSKI, S. *Inductive logic programming - techniques and applications*. Ellis Horwood series in artificial intelligence. Ellis Horwood, 1994. ISBN: 978-0-13-457870-5.
- [30] RAEDT, L. D. “Logical Settings for Concept-Learning”, *Artif. Intell.*, v. 95, n. 1, pp. 187–201, 1997.
- [31] BLOCKEEL, H. *Top-down Induction of First order Logical Decision Trees*. PhD thesis, Katholieke Universiteit Leuven.
- [32] RAEDT, L. D., DZEROSKI, S. “First-Order jk -Clausal Theories are PAC-Learnable”, *Artif. Intell.*, v. 70, n. 1-2, pp. 375–392, 1994.

- [33] WROBEL, S., DZEROSKI, S. “The ILP description learning problem: Towards a general model-level definition of data mining in ILP”. 1995.
- [34] QUINLAN, J. R. “Learning logical definitions from relations”, *MACHINE LEARNING*, v. 5, pp. 239–266, 1990.
- [35] MUGGLETON, S. “Inverse Entailment and Progol”, *New Generation Computing, Special issue on Inductive Logic Programming*, v. 13, n. 3-4, pp. 245–286, 1995.
- [36] KRAMER, S. “Structural Regression Trees”. In: *Proceedings of the Thirteenth National Conference on Artificial Intelligence - Volume 1, AAAI’96*, pp. 812–819. AAAI Press, 1996. ISBN: 0-262-51091-X.
- [37] KARALIČ, A., BRATKO, I. “First Order Regression”, *Mach. Learn.*, v. 26, n. 2-3, pp. 147–176, mar. 1997. ISSN: 0885-6125.
- [38] BLOCKEEL, H., DE RAEDT, L. “Top-down Induction of First-order Logical Decision Trees”, *Artif. Intell.*, v. 101, n. 1-2, pp. 285–297, maio 1998. ISSN: 0004-3702.
- [39] BELLODI, E., RIGUZZI, F. “Structure learning of probabilistic logic programs by searching the clause space”, *TPLP*, v. 15, n. 2, pp. 169–212, 2015.
- [40] RIGUZZI, F. “Learning Logic Programs with Annotated Disjunctions”. In: *ILP*, v. 3194, *Lecture Notes in Computer Science*, pp. 270–287. Springer, 2004.
- [41] KHOSRAVI, H., BINA, B. “A Survey on Statistical Relational Learning”. In: Farzindar, A., Kešelj, V. (Eds.), *Advances in Artificial Intelligence*, pp. 256–268, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg. ISBN: 978-3-642-13059-5.
- [42] DE RAEDT, L., KERSTING, K. “Probabilistic Logic Learning”, *SIGKDD Explor. Newsl.*, v. 5, n. 1, pp. 31–48, jul. 2003. ISSN: 1931-0145.
- [43] DE RAEDT, L., KERSTING, K. “Probabilistic Inductive Logic Programming”. Springer-Verlag, cap. Probabilistic Inductive Logic Programming, pp. 1–27, Berlin, Heidelberg, 2008. ISBN: 3-540-78651-1, 978-3-540-78651-1.
- [44] SATO, T. “A Statistical Learning Method for Logic Programs with Distribution Semantics”. In: *Logic Programming, Proceedings of the Twelfth International Conference on Logic Programming, Tokyo, Japan, June 13-16, 1995*, pp. 715–729, 1995.

- [45] FIERENS, D., DEN BROECK, G. V., RENKENS, J., et al. “Inference and learning in probabilistic logic programs using weighted Boolean formulas”, *TPLP*, v. 15, n. 3, pp. 358–401, 2015.
- [46] WARREN, D. H. D., PEREIRA, L. M., PEREIRA, F. “Prolog - the Language and Its Implementation Compared with Lisp”, *SIGPLAN Not.*, v. 12, n. 8, pp. 109–115, ago. 1977. ISSN: 0362-1340.
- [47] SATO, T., KAMEYA, Y. “PRISM: a language for symbolic-statistical modeling”. In: *In Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI’97)*, pp. 1330–1335, 1997.
- [48] SATO, T., KAMEYA, Y. “Parameter Learning of Logic Programs for Symbolic-statistical Modeling”, *J. Artif. Int. Res.*, v. 15, n. 1, pp. 391–454, dez. 2001. ISSN: 1076-9757.
- [49] POOLE, D. L. “Exploiting the Rule Structure for Decision Making within the Independent Choice Logic”, *CoRR*, v. abs/1302.4978, 2013.
- [50] POOLE, D. “The Independent Choice Logic and Beyond”. In: *Probabilistic Inductive Logic Programming - Theory and Applications*, pp. 222–243, 2008.
- [51] RIGUZZI, F. “SLGAD Resolution for Inference on Logic Programs with Annotated Disjunctions”, *Fundam. Inform.*, v. 102, n. 3-4, pp. 429–466, 2010.
- [52] RAEDT, L. D., KIMMIG, A., TOIVONEN, H. “ProbLog: A Probabilistic Prolog and Its Application in Link Discovery.” In: Veloso, M. M. (Ed.), *IJCAI*, pp. 2462–2467, 2007.
- [53] PEARL, J. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. San Francisco, CA, USA, Morgan Kaufmann Publishers Inc., 1988. ISBN: 0-934613-73-7.
- [54] FIERENS, D., VAN DEN BROECK, G., RENKENS, J., et al. “Inference and learning in probabilistic logic programs using weighted Boolean formulas”, *Theory and Practice of Logic Programming*, v. 15, n. 3, pp. 358–401, 2015.
- [55] DE RAEDT, L., THON, I. “Probabilistic rule learning”. v. 6489, pp. 47–58. Paolo, Frasconi, Springer Verlag, 2010. ISBN: 978-3-642-21294-9.
- [56] DE RAEDT, L., DRIES, A., THON, I., et al. “Inducing Probabilistic Relational Rules from Probabilistic Examples”. In: *Proceedings of the 24th International Conference on Artificial Intelligence, IJCAI’15*, pp. 1835–1843. AAAI Press, 2015. ISBN: 978-1-57735-738-4.

- [57] KOLLER, D., FRIEDMAN, N. *Probabilistic Graphical Models: Principles and Techniques - Adaptive Computation and Machine Learning*. The MIT Press, 2009. ISBN: 0262013193, 9780262013192.
- [58] LOWD, D., DOMINGOS, P. “Efficient Weight Learning for Markov Logic Networks”. v. 4702, pp. 200–211, 09 2007.
- [59] KOK, S., DOMINGOS, P. “Learning the Structure of Markov Logic Networks”. In: *Proceedings of the 22Nd International Conference on Machine Learning*, ICML '05, pp. 441–448, New York, NY, USA, 2005. ACM. ISBN: 1-59593-180-5.
- [60] RICHARDS, B. L., MOONEY, R. J. “Automated Refinement of First-Order Horn-Clause Domain Theories”, *Machine Learning*, v. 19, n. 2, pp. 95–131, 1995.
- [61] DUBOC, A. L., PAES, A., ZAVERUCHA, G. “Using the bottom clause and mode declarations in FOL theory revision from examples”, *Machine Learning*, v. 76, n. 1, pp. 73–107, 2009.
- [62] PAES, A., ZAVERUCHA, G., COSTA, V. S. “On the use of stochastic local search techniques to revise first-order logic theories from examples”, *Machine Learning*, v. 106, n. 2, pp. 197–241, 2017.
- [63] PAES, A., ZAVERUCHA, G., COSTA, V. S. “Revising First-Order Logic Theories from Examples Through Stochastic Local Search”. In: *Inductive Logic Programming, 17th International Conference, ILP 2007, Corvallis, OR, USA, June 19-21, 2007, Revised Selected Papers*, pp. 200–210, 2007.
- [64] RUDER, S. *Neural Transfer Learning for Natural Language Processing*. PhD thesis, National University of Ireland, Galway, 2019.
- [65] CECI, M., APPICE, A., BARILE, N., et al. “Transductive Learning from Relational Data”. In: *Proceedings of the 5th International Conference on Machine Learning and Data Mining in Pattern Recognition*, MLDM '07, pp. 324–338, Berlin, Heidelberg, 2007. Springer-Verlag. ISBN: 978-3-540-73498-7.
- [66] VAPNIK, V. N. *Statistical Learning Theory*. Wiley-Interscience, 1998.
- [67] ARNOLD, A., NALLAPATI, R., COHEN, W. W. “A Comparative Study of Methods for Transductive Transfer Learning”. In: *Proceedings of the Seventh IEEE International Conference on Data Mining Workshops*,

ICDMW '07, pp. 77–82, Washington, DC, USA, 2007. IEEE Computer Society. ISBN: 0-7695-3033-8.

- [68] III, H. D., MARCU, D. “Domain Adaptation for Statistical Classifiers.” *J. Artif. Intell. Res.*, v. 26, pp. 101–126, 2006.
- [69] ZADROZNY, B. “Learning and Evaluating Classifiers Under Sample Selection Bias”. In: *Proceedings of the Twenty-first International Conference on Machine Learning*, ICML '04, pp. 114–, New York, NY, USA, 2004. ACM. ISBN: 1-58113-838-5.
- [70] SHIMODAIRA, H. “Improving predictive inference under covariate shift by weighting the log-likelihood function”, *Journal of Statistical Planning and Inference*, v. 90, n. 2, pp. 227–244, out. 2000.
- [71] CARUANA, R. “Multitask Learning”, *Mach. Learn.*, v. 28, n. 1, pp. 41–75, jul. 1997. ISSN: 0885-6125.
- [72] RAINA, R., BATTLE, A., LEE, H., etal. “Self-taught Learning: Transfer Learning from Unlabeled Data”. In: *Proceedings of the 24th International Conference on Machine Learning*, ICML '07, pp. 759–766, New York, NY, USA, 2007. ACM. ISBN: 978-1-59593-793-3.
- [73] DAI, W., YANG, Q., XUE, G.-R., etal. “Self-taught Clustering”. In: *Proceedings of the 25th International Conference on Machine Learning*, ICML '08, pp. 200–207, New York, NY, USA, 2008. ACM. ISBN: 978-1-60558-205-4.
- [74] WANG, Z., SONG, Y., ZHANG, C. “Transferred Dimensionality Reduction”. In: *ECML/PKDD*, 2008.
- [75] JIANG, J., ZHAI, C. “Instance Weighting for Domain Adaptation in NLP.” In: Carroll, J. A., van den Bosch, A., Zaenen, A. (Eds.), *ACL*. The Association for Computational Linguistics, 2007.
- [76] ARGYRIOU, A., EVGENIOU, T., PONTIL, M. “Multi-Task Feature Learning”. In: Schölkopf, B., Platt, J. C., Hoffman, T. (Eds.), *Advances in Neural Information Processing Systems 19*, MIT Press, pp. 41–48, 2007.
- [77] LAWRENCE, N. D., PLATT, J. C. “Learning to Learn with the Informative Vector Machine”. In: *Proceedings of the Twenty-first International Conference on Machine Learning*, ICML '04, pp. 65–, New York, NY, USA, 2004. ACM. ISBN: 1-58113-838-5.

- [78] BONILLA, E. V., CHAI, K. M., WILLIAMS, C. “Multi-task Gaussian Process Prediction”. In: Platt, J. C., Koller, D., Singer, Y., et al. (Eds.), *Advances in Neural Information Processing Systems 20*, Curran Associates, Inc., pp. 153–160, 2008.
- [79] HECKERMAN, D., CHICKERING, D. M., MEEK, C., et al. “Dependency Networks for Inference, Collaborative Filtering, and Data Visualization”, *J. Mach. Learn. Res.*, v. 1, pp. 49–75, set. 2001. ISSN: 1532-4435.
- [80] NEVILLE, J., JENSEN, D. D. “Relational Dependency Networks”, *Journal of Machine Learning Research*, v. 8, pp. 653–692, 2007.
- [81] FRIEDMAN, N., GETOOR, L., KOLLER, D., et al. “Learning Probabilistic Relational Models”. In: *Proceedings of the 16th International Joint Conference on Artificial Intelligence - Volume 2*, IJCAI’99, pp. 1300–1307, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc.
- [82] TASKAR, B., ABBEEL, P., KOLLER, D. “Discriminative Probabilistic Models for Relational Data”. In: *Proceedings of the Eighteenth Conference on Uncertainty in Artificial Intelligence*, UAI’02, pp. 485–492, San Francisco, CA, USA, 2002. Morgan Kaufmann Publishers Inc. ISBN: 1-55860-897-4.
- [83] NEVILLE, J., JENSEN, D., FRIEDLAND, L., et al. “Learning Relational Probability Trees”. In: *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD ’03, pp. 625–630, New York, NY, USA, 2003. ACM. ISBN: 1-58113-737-0.
- [84] NEVILLE, J., JENSEN, D. D., GALLAGHER, B. “Simple Estimators for Relational Bayesian Classifiers”. In: *ICDM*, pp. 609–612. IEEE Computer Society, 2003.
- [85] DIETTERICH, T. G., ASHENFELTER, A., BULATOV, Y. “Training Conditional Random Fields via Gradient Tree Boosting”. In: *Proceedings of the Twenty-first International Conference on Machine Learning*, ICML ’04, pp. 28–, New York, NY, USA, 2004. ACM. ISBN: 1-58113-838-5.
- [86] GUTMANN, B., KERSTING, K. “TildeCRF: Conditional Random Fields for Logical Sequences”. In: *Machine Learning: ECML 2006, 17th European Conference on Machine Learning, Berlin, Germany, September 18-22, 2006, Proceedings*, pp. 174–185, 2006.
- [87] NATARAJAN, S., KHOT, T., KERSTING, K., et al. “Boosting Relational Dependency networks”. In: *In Inductive Logic Programming*, 2010.

- [88] FRIEDMAN, J. H. “Greedy Function Approximation: A Gradient Boosting Machine”, *Annals of Statistics*, v. 29, pp. 1189–1232, 2000.
- [89] OURSTON, D., MOONEY, R. J. “Changing the Rules: A Comprehensive Approach to Theory Refinement”. In: *Proceedings of the Eighth National Conference on Artificial Intelligence (AAAI-90)*, pp. 815–820, Boston, MA, July 1990.
- [90] BILENKO, M., MOONEY, R. J. “Adaptive Duplicate Detection Using Learnable String Similarity Measures”. In: *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '03*, pp. 39–48. ACM, 2003.
- [91] KHOSRAVI, H., SCHULTE, O., HU, J., et al. “Learning compact Markov logic networks with decision trees”, *Machine Learning*, v. 89, n. 3, pp. 257–277, 2012. ISSN: 08856125.
- [92] MIHALKOVA, L., MOONEY, R. J. “Bottom-Up Learning of Markov Logic Network Structure”. In: *Proceedings of 24th International Conference on Machine Learning (ICML-2007)*, 2007.
- [93] CARLSON, A., BETTERIDGE, J., KISIEL, B., et al. “Toward an Architecture for Never-ending Language Learning”. In: *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence, AAAI'10*, pp. 1306–1313. AAAI Press, 2010.