WORD EMBEDDINGS-BASED TRANSFER LEARNING FOR BOOSTED RELATIONAL DEPENDENCY NETWORKS

Thais Luca Marques de Almeida

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
Dezembro de 2021

WORD EMBEDDINGS-BASED TRANSFER LEARNING FOR BOOSTED RELATIONAL DEPENDENCY NETWORKS

Thais Luca Marques de Almeida

DISSERTAÇÃO SUBMETIDA AO CORPO DOCENTE DO INSTITUTO ALBERTO LUIZ COIMBRA DE PÓS-GRADUAÇÃO E PESQUISA DE ENGENHARIA 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.

Orientadores: Gerson Zaverucha
Aline Marins Paes Carvalho

Aprovada por: Prof. Gerson Zaverucha
Prof. Aline Marins Paes Carvalho
Prof. Daniel Ratton Figueiredo
Prof. Fabio Gagliardi Cozman

RIO DE JANEIRO, RJ – BRASIL
DEZEMBRO DE 2021

Almeida, Thais Luca Marques de

Word Embeddings-based Transfer Learning for Boosted Relational Dependency Networks/Thais Luca Marques de Almeida. – Rio de Janeiro: UFRJ/COPPE, 2021.

XIII, 62 p.: il.; 29, 7cm.

Orientadores: Gerson Zaverucha

Aline Marins Paes Carvalho

Dissertação (mestrado) – UFRJ/COPPE/Programa de Engenharia de Sistemas e Computação, 2021.

Referências Bibliográficas: p. 56 – 62.

1. Transfer learning. 2. Statistical relational learning. 3. Word embeddings. 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 aos meus pais que sempre me incentivaram a fazer o que eu gosto e por sempre me apoiarem nas minhas decisões.

Agradecimentos

Primeiro a Deus que é a fonte de toda a minha força para encarar os desafios que surgiram ao longo dos últimos dois anos.

Aos meus pais, Angela e Carlos, que sempre me incentivaram a estudar e por terem topado a minha loucura de pedir demissão para me dedicar ao mestrado. Por todo o carinho e por terem me ensinado a ter coragem para seguir em frente.

À minha irmã Thati, dona de um senso de humor incomparável que tornou tudo mais fácil nestes tempos de pandemia. Por sempre ter ficado do meu lado, me fazendo rir e me dando força.

Aos meus queridos orientadores, Gerson e Aline, por todo o conhecimento passado e por serem sempre tão compreensivos e pacientes. Ao professor Gerson pela oportunidade de estudar na COPPE. À professora Aline por ter aceitado me orientar. Obrigada por terem confiado no meu potencial, pela disponibilidade para as reuniões e pelos ótimos conselhos durante a realização do trabalho.

À Cristiane, que cuidou de mim nesses últimos três anos, ouvindo minhas histórias, me ajudando a trabalhar minhas inseguranças e por todos os bons conselhos compartilhados toda semana.

Aos professores Daniel Ratton e Fábio Cozman pela participação da banca e pelos comentários que contribuíram muito com o trabalho.

Aos meus amigos que sempre tornaram tudo mais leve e divertido. Principalmente à Leticia, minha amiga e parceira de mestrado com quem pude dividir os estudos e as dúvidas.

À Universidade Federal Fluminense que me emprestou o cluster para execução dos experimentos. Ao Carlos pelo suporte com o Oscar e disponibilidade em tirar minhas dúvidas. Ao Francisco pela ajuda em religar o cluster do LabIA que infelizmente não sobreviveu à pandemia. Ao Rodrigo pela disposição em ajudar com as dúvidas que apareceram ao continuar o seu trabalho.

Aos professores do Programa de Engenharia de Sistema e Computação que foram essenciais nessa caminhada. Agradeço também à equipe administrativa sempre muito solícita.

À Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) que financiou 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.)

EMBEDDINGS DE PALAVRAS PARA TRANSFERÊNCIA DE APRENDIZADO DE REDES DE DEPENDÊNCIA RELACIONAL COM BOOSTING

Thais Luca Marques de Almeida

Dezembro/2021

Orientadores: Gerson Zaverucha
Aline Marins Paes Carvalho

Programa: Engenharia de Sistemas e Computação

Algoritmos de aprendizado de máquina têm obtido sucesso nas mais diversas áreas de aplicação. Porém, os métodos tradicionais assumem dados independentes e identicamente distribuídos (i.i.d.), desprezando a estrutura relacional dos dados, que contém informações cruciais sobre como objetos participam de relações e eventos. Dentre os algoritmos de aprendizado de máquina, os modelos de aprendizado estatístico consistem em uma representação concisa das dependências probabilísticas entre atributos de um objeto. O aprendizado estatístico relacional estende aprendizado estatístico para representar e aprender a partir de dados contendo diferentes objetos e como estes se relacionam. Apesar de não seguirem a suposição i.i.d., também assumem que dados de treinamento e teste seguem a mesma distribuição. Para lidar com cenários em que os dados têm diferentes distribuições, surgiu a transferência de aprendizado, que consiste em usar o conhecimento adquirido em uma ou mais tarefas já resolvidas como um ponto de partida para resolver uma nova tarefa. Para aplicar transferência de aprendizado em aprendizado estatístico relacional, o primeiro desafio é como transferir a estrutura, mapeando o vocabulário de um domínio de origem para um domínio de destino. Nesta dissertação, propomos o TransBoostler, que utiliza vetores de palavras pré-treinados para mapear vocabulários, uma vez que os nomes dos predicados normalmente tem uma conotação semântica que pode ser mapeada para um modelo de espaço vetorial. Após a transferência, aplica-se revisão de teoria para adaptar o modelo mapeado aos novos dados de treinamento. Durante os experimentos, o TransBoostler realizou com êxito a tarefa de transferir árvores entre domínios com desempenho igual ou superior a trabalhos anteriores, e com redução no tempo de treinamento para a maioria dos cenários investigados.

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

WORD EMBEDDINGS-BASED TRANSFER LEARNING FOR BOOSTED RELATIONAL DEPENDENCY NETWORKS

Thais Luca Marques de Almeida

December/2021

Advisors: Gerson Zaverucha
Aline Marins Paes Carvalho

Department: Systems Engineering and Computer Science

Machine learning algorithms have proven to be a great asset in different applications. However, traditional machine learning methods assume data is independent identically distributed (i.i.d.) and despises the relational structure of the data, which contains crucial information about how objects participate in relationships and events. Statistical machine learning models are a concise representation of probabilistic dependencies among the attributes of an object. Statistical Relational Learning (SRL) extends statistical learning to represent and learn from data with several objects and their relations. SRL models do not suppose data to be i.i.d. but, as traditional machine learning models, also assume training and testing data are sampled from the same distribution. Transfer learning has emerged as an essential technique to handle scenarios where such an assumption does not hold, as it relies on leveraging the knowledge acquired in one or more learning tasks as a starting point to solve a new task. When employing transfer learning to SRL, the primary challenge is to transfer the learned structure, mapping the vocabulary from a source domain to a different target domain. In this dissertation, we propose TransBoostler, which uses pre-trained word embeddings to guide the mapping as the name of a predicate usually has a semantic connotation that can be mapped to a vector space model. After transferring, TransBoostler employs theory revision to adapt the mapped model to the target data. In the experimental results, TransBoostler has successfully transferred trees from a source to a different target domain. It performs equal or better than previous works and requires less training time for most of the investigated scenarios.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

Machine Learning is a subfield of Artificial Intelligence concerned with constructing computer programs able to perform a task by experience [4]. Machine learning algorithms have proven to be a great asset in different applications such as text mining, computer vision, speech recognition, and others [5–7]. Traditional machine learning methods assume data is independent and identically distributed (i.i.d.). In this way, data is represented in a tabular format using attribute-value pairs. Each example corresponds to a single row or tuple, and each feature or attribute to a single column. This type of representation despises the relational structure of the data, which contains crucial information about how objects participate in relationships and events [8]. Most real-world data is relational and consists of different types of entities characterized by a different set of attributes [9].

Statistical Relational Learning (SRL) combines elements from statistical and probabilistic modeling to relational learning aiming at representing, reasoning, and learning in domains with complex relational and rich probabilistic structures [10]. Thus, the input to an SRL learning algorithm is often just a single and richly connected instance and not a sequence of i.i.d. observations as for traditional machine learning methods. SRL has succeeded in many real-world applications as real data also requires handling uncertainty from noise and incomplete information like misspellings and occlusions. As most machine learning models, SRL models also assume training and testing data must belong to the same feature space and are sampled from the same distribution. If those distributions differ from each other, a new model must be trained using newly collected data.

To address the existence of training and testing data from different distributions, Transfer Learning [1] has emerged as an important technique given that collecting new data can be a costly or even impossible task. Transfer learning has recently gained much interest from researchers due to its success in Deep Learning applications [6]. Based on how people can intelligently apply knowledge learned previously to solve new problems, transfer learning relies on leveraging the knowledge acquired

in one or more learning tasks and domains to achieve a good initial performance for solving a new task. One example is the classification of emails as *spam* or *not spam*. One needs to collect a lot of labeled emails from a group of users to train a classifier. If a new email user with an email distribution different from the first set shows up, we might adapt the learned model for this user. Transfer learning cares most about the target tasks rather than learning all of the source and target tasks simultaneously. Furthermore, it also targets at reducing the amount of time it takes to learn a model from scratch [11]. More importantly, applying transfer learning to SRL models admits training and testing domains to differ in distributions as it has successfully been verified in previous works [3, 12, 13]. However, relational learning differs from function-based and traditional machine learning methods since in the former data usually has a rich vocabulary composed of classes, objects, their properties and relationships [14]. Therefore, the challenge of applying transfer learning to SRL models is primarily how to transfer the learned structure, mapping the vocabulary from a source domain to the most appropriate objects, properties, and relations in a different target domain. For example, suppose a model built upon a movie domain defining a relation between an actor working under a director. Now, suppose one wants to learn the concept of a student being advised by a professor in an academic domain. Arguably, those relations in both domains have a similar semantic, and learning the *advised by* relation could benefit from the concept learned for describing the *worked under* relation. However, to transfer the concept from the source movie domain to the target university domain, there are other predicates that need to be mapped, besides the main ones. For example, the movie domain has actor, director, movie, properties that could be mapped to student, professor, publication predicates, to name a few.

TreeBoostler [3], a system that employs transfer learning to the Relational Dependency Boosting (RDN-Boost) framework [2], recursively tries to transfer nodes from source relational regression trees to build target relational regression trees. It tries every possible mapping from a source predicate and chooses the best mapping using weighted variance as the decision criterion. Trying every possible mapping can be costly and time-consuming. Thus, devising other more efficient mechanisms for mapping the vocabulary is vital in Statistical Relational learning methods.

In this dissertation, we propose to use pre-trained word embeddings [15] to guide the mapping as the name of the predicates usually have a semantic connotation that can be mapped to a Vector Space Model (VSM). The mechanism proposed is named as TransBoostler. As TreeBoostler, it also focuses on transferring Boosted Relational Dependency Networks (RDNs) but it uses pre-trained word vector representations of predicates for mapping. As presented in [2], boosting RDNs has superior performance when compared to traditional SRL approaches. TransBoostler maps the

predicates that appear in trees learned in the source domain to the most similar predicates in a target domain. Thus, this approach leads to a richer mapping as it takes advantage of the context of embeddings to choose mappings which also reduces the searching space.

Nevertheless, only transferring vocabulary may result in possible faults that can prevent theories from predicting examples correctly [13]. As TreeBoostler, TransBoostler also includes a Theory Revision [16] component to propose modifications in order to count on predicates from the target domain that were not mapped to any predicate from the source domain. Furthermore, the same concept can be expressed in different ways in the VSM. So the revision component may accommodate modifications pointed out by the target training data. In the context of relational regression trees, theory revision includes modifying the trees by adding or removing branches in specific locations. Those locations are selected according to the performance of the current tree to cover or not positive and negative examples.

1.1 Research Questions

This dissertation aims at answering the following research questions, regarding similarity-based transfer learning and baselines:

Q1 Does TransBoostler learn more accurate models than the baselines?

Q2 Can TransBoostler transfer theories by relying on word embeddings similarity?

Q3 How important is revising the theory when transferring relies on word embeddings-based similarity?

Q4 Is the mapping by similarity approach faster than the baselines?

Q5 Does TransBoostler perform better than the baselines with increasing amounts of examples in the target data?

In order to answer these questions, we evaluated TransBoostler in real-world relational datasets. TransBoostler trains on one single fold and test the remaining folds to simulate the scenario of few data available. Then, we compare the performance of TransBoostler for different amounts of target data using traditional cross-validation methodology. We also tested mapping by similarity using four different similarity metrics. Our results demonstrate that the proposed algorithm can successfully transfer learned theories across different domains by mapping predicates using similarity (**Q2**). However, mapping by similarity can impair performance depending on the source and target domains, and the amount of data available (**Q1** and **Q5**).

For some pairs of datasets, TransBoostler proved to be faster than at least one of the baselines. However, for others, it is no longer faster than learning from scratch (**Q4**). Lastly, results show that TransBoostler cannot learn accurate models by just transferring the structure. Thus, for most of the experiments, revising the structure is important to improve the theories (**Q3**).

1.2 Contributions

The contributions of this dissertation include:

- A word embeddings similarity-based transfer learning algorithm that builds target trees based on the structure learned in previously relational tasks. It takes advantage of the context of predicates and performs predicate mapping by similarity, which has good performance and can be less time-consuming;
- A mapping component that can be applied to different and more general relational models, which is the main contribution of this dissertation;
- An accepted paper at the 30th International Conference on Inductive Logic Programming. The paper "Mapping Across Relational Domains for Transfer Learning with Word Embeddings-based Similarity" [17] was selected as the recipient of the best student paper award for the conference track.

1.3 Outline

The remainder of this dissertation is organized as follows:

Chapter 2, introduces necessary background to understand the contents of this work. It is given a brief overview of First-Order Logic, Inductive Logic Programming, and Statistical Relational Learning. We describe the algorithm for boosting RDNs along with concepts for understanding Relational Dependency Networks, Functional Gradient Boosting, and Relational Regression Trees. Secondly, we review Word Embeddings, the fast-Text model, and the similarity metrics used in this work. Lastly, we present some related work and compare the literature with our proposed method.

Chapter 3 presents the proposed algorithm named TransBoostler and its process of transference. We propose two approaches to perform mapping by similarity using pre-trained word embeddings: (1) following the order in which predicates appear in the source structure, so predicates closer to the root are the first to be mapped and; (2) mapping is performed by following an ordered list of similarities between pairs

of source and target predicates. In this approach, the order in which each predicate appears in the source structure does not matter.

Chapter 4 presents the experimental results of TransBoostler for different datasets. We compare results with previous work and learning from scratch in the target domain in two experiments for each mapping approach. The first experiment simulates a transfer learning environment with limited target data. The second considers a scenario with increasing amounts of target data.

Finally, Chapter 5 presents conclusions and some future work directions.

Chapter 2

Background Knowledge

In this chapter, we introduce an overview of concepts used to build this work. First, we describe the concepts related to relational learning by presenting First-order logic, Inductive Logic Programming, and an overview of Statistical Relational Learning. Then, we describe the Relational Dependency Boosting (RDN-Boost) framework along with Relational Dependency Networks, Functional Gradient Boosting, and Relational Regression Trees. Third, we introduce an overview of Transfer Learning. Finally, we introduce Word Embeddings, the fast-Text model, and the similarity metrics applied for mapping predicates. Pre-processing tools used in this dissertation are briefly described as they are mentioned in Section 3.2.

2.1 Relational Learning

Traditional machine learning methods such as decision trees, artificial neural networks, and linear models expect inputs in a tabular format. In this type of representation, each example corresponds to a single row or tuple, and each feature or attribute to a single column. This type of representation considers attribute-value pairs and assumes independent and identically distributed (i.i.d.) entities.

These techniques are limited from a knowledge representation perspective, which is essentially propositional (based on boolean or propositional logic). Propositional representation despises the relational structure of the data, which contains crucial information about how objects participate in relationships and events [8]. Most real-world data is relational, and consist of different types of entities characterized by different sets of attributes [9] such as chemical databases like Yeast [18]. Then, exploring the structure of relational data allows finding solutions to more general and complex problems [8].

Relational representation arises as a more expressive knowledge representation for learning because it can represent domains for multiple entities and the relationships among them. Relational datasets store data across multiple tables, where each

table represents different types of entities and how these entities may relate to each other. In addition, dealing with real data also requires the ability to handle uncertainly that can arise on many levels. Real-world data can be noisy and/or contain incomplete information like occlusions and misspellings. Relational learning tasks require sophisticated treatment of uncertainly at multiple levels of representation.

2.1.1 First-Order Logic

A robust way to represent relational data is using First-Order Logic (FOL). FOL is a formal language that aims to knowledge representation and reasoning in Artificial Intelligence [19]. It is an extension of propositional logic to represent objects, their properties, and how these objects relate to each other [20]. It also assumes that certain relations between objects may or may not hold. Domains are represented by logical facts containing predicates and terms. Logical facts are statements, and a set of logical facts form a *knowledge base*. Knowledge bases can also contain rules.

Constants are used to represent objects from the real world. Variables are terms to be substitute by constants or function symbols to answer questions about which constants relate to each other. Predicates represent relations between objects in the domain. We follow the Prolog [21] syntax. Variables' names start with capital letters and predicates and constants are in lowercase. The example *publication(title, jane)* is a logical fact that can be used to represent the relation between a published material identified by *title* and the person named as *jane* that wrote that material. Both are objects of the real world.

A relation is defined by a set of tuples of objects that satisfies it. In the former example, *publication* is the name of the predicate while *title* and *jane* are constants representing the entities of the domain. The name does not matter formally, but it is important for readability to a clear interpretation of symbols. Arguments of a predicate are associated with a type and predicates have an *arity*, i.e., the size of the tuple of arguments to represent a relation. In our example, the first argument is associated with the type *title* while the second is associated with the type *person*. Since we have two arguments, we say this predicate is of arity two or its a binary relation. Predicates can also represent properties of objects. Two examples are *actor* and *director* to distinguish if a person is an actor or a director (or both). As *actor* and *director* have one single argument of type *person*, we say their arity is one or its a unary relation. Predicates are usually referred to as *name/n*, where *n* is their arity.

Some relations are *functional*, which means a given object is related to exactly one other object by the relation. Taking the relation *cosine* as example, any angle has one and only number that is its cosine. It is a mapping of a set of tuples in

which every input is related to a unique output. Back to the example, the function symbol *cosine* gives the cosine of every possible angle of interest. Function symbols are used to refer to objects without using their names.

A term can be a variable, a constant or a function symbol applied to terms. Terms represent objects of the real world. If $t_1, ..., t_n$ are terms, and f a function symbol of arity n, then we say $f(t_1, ..., t_n)$ is also a term [22]. An atom is a predicate applied to terms, i.e., a predicate symbol followed by a list of terms as its arguments. Our example *publication(title, jane)* is an atom that states that *title* was published by *jane*. The atom may be *true* or *false*, depending on if it holds or not a relation in the real world. Atoms that assert a relationship among constants are called ground atoms (e.g. *publication(title, person)*). A literal can be an atom or a negated atom. If the atom is *true*, then the negated atom is *false* and vice-versa. In our example, $\neg$ *publication(title, jane)* states that *title* was not published by *jane*.

Atoms are connected to build formulas. A single atom is already a formula and connectives can be used to build complex formulas. There are five logical connectives. Considering two formulas P and Q, the connectives are:

- $\neg$ (not) means $\neg P$ is the negation of P. It is the only connective that operates on a single atom. It states that $\neg P$ is *true* whenever P is *false*;
- $\wedge$ (and) states that (P $\wedge$ Q) is *true* whenever both atoms P and Q are *true*. Then, $P \wedge Q$ is called a *conjunction*. If one of the literals is *false*, then the conjunction is *false*;
- $\vee$ (or) states that $(P \vee Q)$ is *true* if at least one of the literals, P or Q, is *true*. This construction is called a disjunction;
- $\rightarrow$ or $\leftarrow$ (implies). A formula such as $P \rightarrow R$ is an implication or conditional. That means R is a conclusion or consequent of its premise P. Then, $P \rightarrow R$ is *true* whenever R is *true* or both P and R are *false*;
- $\leftrightarrow$ (equivalent) states that $P \leftrightarrow$ Q is *true* whenever both P and Q have the same logical value. This type of construction can also be called biconditional.

FOL has two standard quantifiers: the *universal* quantifier ($\forall$) and the *existential* quantifier ($\exists$). Quantifiers are used to express properties or concepts about the entire collection of objects in the real world. We can express the information *"Socrates is a human then he is a mortal"* using FOL:

$$human(socrates) \rightarrow mortal(socrates)$$

If we want to say that every human is mortal, we can use the *universal* qualifier:

$$\forall X \; human(X) \rightarrow mortal(X)$$

The formula above is ***true*** if and only if all sentences obtained by substituting X for a constant are ***true***, i.e., if the implication is ***true*** for all objects in the universe.

The information "*There is a prime number that it is even*" can also be expressed using FOL. By using the ***existential*** qualifier, we can say that exists at least one prime number that is even:

$$\exists X \; prime(X) \wedge even(X)$$

The formula above will be ***true*** if $prime(X) \wedge even(X)$ is ***true*** for some object in the universe. A disjunction of literals preceded by a universal quantifier is called a clause. It must have a quantifier for each variable presented in the disjunction of literals. A Horn clause is a disjunction of literals with at most one positive literal. A Horn clause is usually written in its implication form and omitting the quantifiers. In the example below, the disjunction *daughter(Y,X)* $\vee$ *son(Y,X)* is called the body of the clause, and *parent* is called the head.

$$parent(X, Y) \leftarrow daughter(Y, X) \vee son(Y, X)$$

A Horn clause with exactly one positive literal is called a definite clause. Where there are no positive literals, the clause is called a negative Horn clause. A fact is a clause whose body is empty, consisting of a single positive literal. We usually do not use arrows while representing facts (e.g. ***parent(james,harry)***).

2.1.2 Inductive Logic Programming

Inductive Logic Programming (ILP) is a subarea of Artificial Intelligence formed at the intersection of Machine Learning and Logic Programming [23]. ILP deals with learning a general theory, given a set of ground atoms as examples to learn a target predicate to infer the value of unseen examples [22]. ILP takes a set of logical facts (knowledge base) into account and deals with two kinds of examples: positives and negatives. Positive examples are the true ones, while relationships that do not hold are called negative examples. Positive and negative examples are given as sets E^+ and E^-, respectively, of ground atoms. Suppose we would like to use ILP to learn the relation *parent(X,Y)*, which states that a person X is the father/mother of a person Y. To learn the *parent* relation, consider the following knowledge base:

$$KB = \begin{cases} father(arthur, ginny) \\ father(james, harry) \\ mother(molly, ginny) \\ mother(lillian, harry) \end{cases}$$

The following facts are our positive examples:

$$E^+ = \begin{cases} parent(arthur, ginny) \\ parent(james, harry) \\ parent(lillian, harry) \end{cases}$$

And the following facts are our negative examples, i.e., relationships that do not exist in the real world:

$$E^- = \begin{cases} parent(ginny, arthur) \\ parent(harry, james) \\ parent(harry, lillian) \end{cases}$$

As we already have the information provided from KB and using the new facts E^+ and E^-, we want to find the relationships:

$$H : parent(X, Y) \leftarrow father(X, Y)$$

$$parent(X, Y) \leftarrow mother(X, Y)$$

where H is our theory, a finite set of clauses, and, together with KB, should cover all the given positive examples presented in E^+ (*completeness*):

$$KB \wedge H \models E^+$$

Hence, H is not a consequence of KB and E^- (*consistency*):

$$KB \wedge H \wedge E^- \not\models \square$$

Consistency and *completeness* together form *correctness*. Then, H is *correct* if it is *complete* and *consistent* [22]. The goal of an ILP system is to find a hypothesis that covers all positive (complete) and none of the negative examples (consistent).

To provide the reader with a general understanding of ILP algorithms and implementations, Algorithm 2.1 presents a generic ILP algorithm. It starts from the predicate we want to learn, set as the head of a rule whose body starts empty. Each step of the algorithm consists of adding a literal to the body of the rule to specialize it in an attempt of making the clause not prove negative examples proved before while still proving the positive examples. This literal can be one of the predicates from the problem statement, the negation of one of the predicates from the problem statement, equality between two bound variables and inequality between two bound variables. Recursive literals are allowed in some ILP systems if they do not cause an infinite regression.

Unfortunately, it might be not possible to find a hypothesis that is both complete and consistent. Therefore, we want to find a hypothesis as close as possible to correct. Thus, the goal is to find the best hypothesis given a space of candidate solutions as ILP can be considered a search problem [23]. ILP systems differ by the direction in which the search starts [22]. Considering how it searches for a new clause, Algorithm 2.1 relies on the *top-down* approach as it starts from an empty theory, the most general possible theory, proving all examples, and specialize it at each step if needed. There is also the *bottom-up* approach, which starts from the most specific theory and generalizes it. ILP systems usually use a quality criterion as the stop criteria. As the goal is to classify unseen examples, one possible quality criterion is accuracy. Algorithms stop searching for better solutions when it reaches the desired percentage of correctly classified objects.

Algorithm 2.1: A generic ILP algorithm.

```
Function GENERIC_ILP_ALGORITHM(KB, E+, E-, target_predicate):
    theory ← {}
    while E+ > 0 do
        clause ← learnNewClause(KB, E+, E-, target_predicate)
        E+ ← E+ \ {positive examples covered by clause}
        theory ← theory ∪ clause
    end
return theory
```

2.1.3 Statistical Relational Learning

ILP has been a proper solution when dealing with relational data. However, the learned rules have a deterministic nature [14]. The objective of manipulating relational data is to reach conclusions about an entity based on the properties of other entities to which the first is related. These conclusions can be reached by finding correlations that are not deterministic by exploring information between links [14].

Besides, dealing with real data also requires the ability to handle uncertainly that can arise on many levels. Real-world data can be noisy and contain incomplete information like occlusions and misspellings. Relational learning tasks require sophisticated treatment of uncertainly at multiple levels of representation [10]. It might have uncertainty about the attributes of objects, an object's type, identity, and relationship membership.

Statistical Relational Learning (SRL) combines elements from statistical and probabilistic modeling to represent and learn in domains with complex relational and rich probabilistic structure [10]. SRL models are a concise representation of probabilistic dependencies among the attributes of different related objects. Representations using SRL can be based on logic or frame-oriented formalisms. Most of its popular tasks are collective classification, linked-based clustering, and link prediction [9]. There are several different SRL formalisms, including Probabilistic Relational Models [24], Relational Dependency Networks [25], Bayesian Logic Programming [26] and Markov Logic Networks [27]. We present a brief review of each model, except for relational dependency networks, which we detail in the next section.

Probabilistic Relational Models (PRMs) are the first successful methods proposed for SRL [9]. It combines logical representation with probabilistic semantics based on directed graphical models. In PRMs, random variables correspond to attributes from different tables, and edges represent correlations between sets. There are two types of PRMs: PRMs that consist of fixed objects and their relationships fixed and uncertainly is only over descriptive attributes of entities and relationships; and PRMs with structural uncertainly in which objects are fixed but uncertainly is over objects to which relationships correspond to. Parameters are learned using the likelihood function which is defined as the probability of the data given the graphical model. To learn the structure, PRMs use greedy algorithms that iteratively modify the structure by adding, removing, or reversing edges to increase the score. The *maximum a posteriori* (MAP) and score functions like Bayesian Information Criterion (BIC) [28] can be used for evaluating different structures. The structure with highest score is chosen as the next candidate.

Bayesian Logic Programming (BLP) is based on Bayesian networks [29]. It uses logic programming to unify Bayesian networks with logic programming. BLPs use Bayesian clauses that use a conditional probability table to present the distribution of the head of the clause conditional on its body [9]. Also, BLPs use combining rules to unite the information on a single literal that is the head of several clauses. BLPs are produced from logical programs. Logical programs consist of sets of clauses. BLPs use Bayesian clauses, which are different from logical clauses because they use a conditional probability table to keep the probability of the head

conditioned to its body. Combining rules are used to compute the conditional probability distribution for the variable that includes the union of all parents. Learning in BLPs is a probabilistic extension of learning in ILP. A score function is used to evaluate how accurate the clauses are and the best match refers to parameters of associated conditional probability distributions that maximize the score function. This score function is based on maximum likelihood. Structure learning follows the same approach of rule learning in ILP systems, i.e., adding and deleting literals, instantiating variables, and using unification of variables on literals or clauses. Several operations are executed simultaneously to speed up the learning procedure.

Markov Logic Networks (MLNs) extend FOL by adding weights for each formula [9]. MLNs are a set of pairs of formulas and their corresponding weights. Every formula is in FOL and weights can be any real number. Each ground in a MLN is represented as a binary node. The value of a node is 1 if the ground atom is *true* and 0 if it is *false*. An edge connects two nodes if the ground predicates appear together in at least one grounding of a formula. The size of the model grows with the number of objects. Parameter learning consists of finding the weight of the formulas, which is equivalent to computing parameters in other models. Weights are computed by maximizing the pseudo-likelihood of the data, which approximates the log-likelihood that is NP-hard to compute. Structure is learned using CLAUDIEN [30] system, for example.

Inference is computational complex and the biggest limitation of most SRL models. Pseudo-likelihood can fail in giving significant results if querying variables are distant in the model. Also, structure learning in SRL can face difficulties of scalability and efficiency due to large datasets. Similar to ILP methods, MLNs' structure learning is not scalable and very inefficient for large datasets. Inference is quicker in RDNs because they approximate joint distributions.

2.2 RDN-Boost

In this section, we introduce the SRL framework used in this work, namely RDN-Boost. RDN-Boost learns a set of relational regression trees using gradient-based boosting. Before describing the algorithm itself, the following subsections give an overview of Relational Dependency Networks and Functional Gradient boosting. Finally it describes the learning algorithm along with Relational Regression Trees.

2.2.1 Relational Dependency Networks

Propositional data record the characteristics of homogeneous and statistically independent objects. Relational data, in contrast, record characteristics of heterogeneous

objects and the relations among these objects [10]. Many machine learning research has focused on "flattened" propositional data, which despises the relational structure of the data and, as a consequence, crucial information. Relational models extend the flat propositional representation of the variables and conditional dependencies among them to relational representation. Also, they remove the assumption of i.i.d. instances assumed by conventional learning techniques.

One of such models are Relational Dependency Networks (RDNs) [25], which are graphical models that have the capacity of expressing and reasoning over dependencies. Also, dealing with real-world data requires handling with autocorrelation, which is a statistical dependency between the values of the same variable on related entities and a ubiquitous characteristic of relational datasets. RDNs allow representing cyclic dependencies which are required to express and exploit autocorrelation during collective inference [25]. In addition, they consist of a simple method for structure learning and parameter estimation, so models are easier to understand and interpret. As RDN extends Dependency Networks, we first introduce the latter.

Dependency Networks (DNs) [31] allows cyclic dependencies as it accepts bidirectional relationships among variables. DNs are an approximate representation of the joint distribution with a set of conditional probability distributions that are learned independently [10]. It encodes probabilistic relationships among a set of variables X in a way that combines characteristics of both undirected and directed graphical models. Dependencies among variables are represented with a bidirected graph $G = (V, E)$ in which dependencies are quantified with a set of conditional probability distributions P. Each node $v_i \in V$ corresponds to an feature $X_i \in X$ and is associated with a conditional probability distribution $P(v_i|\mathbf{Pa}(v_i))$ that gives the probability of the feature given its parents. G contains a directed edge from each parent node v_j to each child node v_i, i.e. $e(v_j, v_i) \in E$ iff $X_j \in Pa(v_i)$, where $Pa(v_i)$ is the set of parents of node v_i, a set of variables that render X_i conditionally independent of the other variables: $P(v_i|V - v_i) = P(v_i|\mathbf{Pa}(v_i))$.

RDNs extend DNs to work with relational data by approximating the joint distribution of a set of random variables as a product of conditional distributions over a ground atom. RDNs consist of a set of *predicates* and *function* symbols that can be grounded given the instantiation of variables. Associated with each predicate Y_i in the domain is a conditional probability distribution $P(Y_i|\mathbf{Pa}(Y_i))$ that defines the distribution over the values of Y_i given its parents' values $\mathbf{Pa}(Y_i)$ [25]. Aggregators such as *count*, *max* and *average* can be used to combine the values of groundings. Figure 2.1 presents an example of RDN for the movies domain. The nodes indicate predicates and the edges probabilistic dependencies between predicates. Note that there are bidirectional relationships between *actor*, *director* and *movie* predicates because if a person has participated in a movie, there is a probability this person

is an actor or a director. Then, learning RDNs corresponds to learn conditional probability distributions. To capture there distributions, NEVILLE and JENSEN [25] use relational probability trees [32] and relational Bayesian classifiers [33].

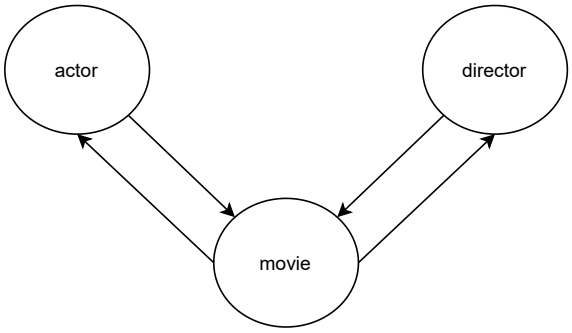

Figure 2.1: Example of an RDN for the movies domain.

2.2.2 Functional gradient boosting of RDNs

Proposed by NATARAJAN *et al.* [2], gradient boosting of RDNs is based on the method proposed by FRIEDMAN [34] called *gradient-tree boosting*, in which potential functions are represented by sums of regression trees that grown stage-wise.

Assuming that training examples are of the form $(\mathbf{x}_i, y_i)$ for $i = 1, \cdots, N$ and $y_i \in \{1, \cdots, K\}$, RDN-Boost is based on gradient-ascent where the learning algorithm starts with an initial potential ψ_0 and iteratively adds gradients Δ_i. After m iterations, the potential is given by

$$\psi_m = \psi_0 + \Delta_1 + \cdots + \Delta_m \tag{2.1}$$

As proposed by FRIEDMAN [34], Δ_m is the functional gradient at episode m given by

$$\Delta_m = \eta_m \times \mathbb{E}_{x,y}\left[\frac{\partial}{\partial_{\psi_{m-1}}} logP(y|\mathbf{x}; \psi_{m-1})\right] \tag{2.2}$$

where η_m is the learning rate. The expectation $\mathbb{E}_{x,y}$ cannot be computed because the joint distribution $P(\mathbf{x}, \mathbf{y})$ is unknown. Then, the functional gradient methods treat data as a surrogate for the joint distribution and compute functional gradients for each training example $(\mathbf{x}_i, y_i)$, instead of computing over the potential function, conditioned on the potential from the previous iteration (ψ_{m-1}):

$$\Delta_m(y_i; \mathbf{x}_i) = \nabla_\psi \sum_i log(P(y_i|\mathbf{x}_i; \psi))|_{\psi_{m-1}} \tag{2.3}$$

As pointed out by DIETTERICH *et al.* [35], functional gradient boosting is the fitting of a regression function $h_m(y, x)$ on the training examples $[(\mathbf{x}_i, y_i), \Delta_m(y_i; \mathbf{x}_i)]$.

$h_m(y, x)$ is not exactly the same as Δ_m but it will point in the same direction assuming that are given enough training examples. Then, the regression function is trained in form of regression tree h_m and fitted to minimize

$$\sum_i [h_m(y_i, \mathbf{x}_i) - \Delta_m(y_i, \mathbf{x}_i)]^2 \tag{2.4}$$

over all examples. It allows learning both the structure and the parameters of RDNs simultaneously. Interactions among variables are introduced only when needed, the algorithm does not consider the potentially large search space, so the number of parameters grows with the number of the training episodes. Also, the algorithm is fast and straightforward to implement, and combining multiple regression trees contributes to avoiding overfitting [35].

2.2.3 Algorithm for learning RDNs

Each conditional probability distribution learned in RDNs can be represented as Relational Regression Trees (RRTs) [32]. Following previous work [36], RDN-Boost uses RRTs to fit the gradient function at every feature in the training examples. Different from classical regression trees such as decision trees, RRTs' inner nodes (or test nodes) are conjunctions of literals. In RRTs, a variable introduced in some node cannot appear in its right sub-tree, so new variables are bound along left-tree paths [36]. Figure 2.2 presents an example of RRT.

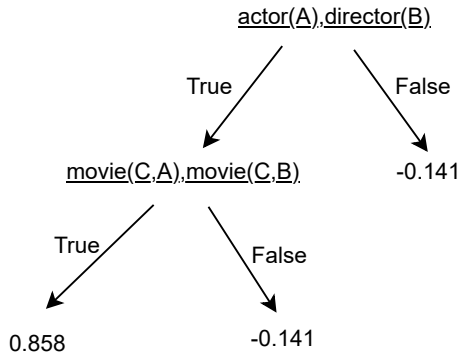

Figure 2.2: Example of RRT to predict if A *workedunder* B.

For RDN-Boost, boosted trees are learned for a given relation and combined in ensemble. Each RRT can be viewed as new features combinations, each one corresponding to each path from the root to a leaf. It implements the regression tree learner named TILDE [37]. The learning algorithm starts with an empty tree and repeatedly searches for the best test for a node according to some splitting criterion such as weighted variance. Similar to classical regression trees, examples in the node are split into *success* and *failure*, according to the test. After every split, this

procedure is recursively applied to obtain its corresponding subtrees. RDN-Boost uses weighted variance as the test criterion.

Back to the example presented in Figure 2.2, the goal is to predict if A *workedunder* B. Each path between the root and leaf can be considered as a clause in a logic program. Then, clauses are evaluated from left to right for a given query and the corresponding regression value is returned. In this example, B is a *director* and A is an *actor* and both worked in the same *movie*, as evaluated by node *movie(C,A),movie(C,B)*. Then, the value returned is 0.858. If A is not an *actor* and/or B is not a *director*, node *actor(A),director(B)* is not satisfied, then the regression value returned is -0.141. Intuitively, if A is not an actor and B is not a director, there is a lower probability of A worked under B. Negative values indicate lower probabilities. Thus, the left-most path of the tree is *actor(A)* $\wedge$ *director(B)* $\wedge$ *movie(C,A)* $\wedge$ *movie(C,B)* $\rightarrow$ *workedunder(A,B)*. The regression tree learner is also able to learn recursive rules by introducing special predicates, such as *recursive_target* for the *target* predicate.

The algorithm proposed by NATARAJAN *et al.* [2] is named RDN-Boost and is presented in Algorithm 2.2. It iterates over all predicates, and, for each predicate k, it generates all examples for the regression tree learner (calling the *FitRelRegressTree* function) to get the new regression tree and updates its model (F_m^k). This process is repeated for a number of iterations m, which is the number of boosted trees. After m steps, the current model F_m^k will approximate the corresponding gradient for the predicate k. Each regression tree serves as the individual components ($\Delta_m(k)$) of the final potential function. The initial potential F_0^1 is usually set to capture the uniform distribution considering all experiments.

The function *GenExamples* generates the examples for the regression-tree learner and takes as input the current predicate index k, the data, and the current model F. It iterates over all examples and computes the probability and the gradient for each one. Computing the probability y_i is done considering all the trees for Y_i. Regression values are computed based on the groundings of the current example, so the gradient is set as the weight of the example. The main algorithm iterates over all examples and learns the potentials for each predicate. The set of regression trees for each predicate forms the structure of the conditional distribution and the set of leaves form the parameters of the conditional distribution.

2.3 Transfer Learning

Machine learning models often assume training and test data are sampled from the same distribution [38]. If distributions differ, a new model must be trained from scratch using newly collected training data. Furthermore, SRL models may suffer

Algorithm 2.2: RDN-Boost: Gradient Tree Boosting for RDN's [2]

Function `TreeBoostForRDNs(Data)`:
  for $1 \geq k \geq K$ **do**
    for $1 \geq m \geq M$ **do**
      $S_k \leftarrow$ GenExamples(k;Data;F^k_{m-1})
      $\Delta_m(k) \leftarrow$ FitRelRegressTree(S_k;L)
      $F^k_m \leftarrow F^k_{m-1} + \Delta_m(k)$
    end
    $P(Y_k = y_k|\mathbf{Pa}(X_k))\ \alpha\ \psi^k$
  end
return
Function `GenExamples(k,Data,F)`:
  $S \leftarrow 0$
  for $1 \geq i \geq N_k$ **do**
    Compute $P(y^i_k = 1|\mathbf{Pa}(x^i_k))$
    $\Delta(y^i_k; x^i_k) \leftarrow I(y^i_k = 1) - P(y^i_k = 1|\mathbf{Pa}(x^i_k))$
    $S \leftarrow S \cup [(x^i_k, y^i_k), \Delta(y^i_k; x^i_k)]$
  end
return S

from the lack of high-quality data instances and a long training time. When there are too many relations, the available ones may be too scarce to learn an accurate model [1].

Transfer Learning [1] has emerged as an important technique when training and testing data differ in distribution. As data can be easily outdated, and newly data can be expensive or impossible to collect, it may be the key to reduce re-calibration effort as a model trained in one time period can be adapted to predict data in a new time period [1, 38]. Besides, as most machine learning models may only succeed when trained using large amounts of data, models may have poor performance in new scenarios. Transfer learning aims at providing machine learning methods with the ability of recognizing knowledge previously learned in a source domain and apply this knowledge to a new model in a target domain. It contributes to improving performance and tends to make learning a new task less time- and data-consuming, as exploiting knowledge learned from one or more previous tasks avoids learning from scratch one specific domain. Most important, it allows domains, tasks, and distributions to differ [38]. It also assists in solving data sparsity and cold start problems in large-scale and online applications [1]. Transfer Learning has attracted researchers due to its success in Deep Learning applications [11]. And it is suitable for relational learning to overcome the reliance on large-high-quality data as it relies on useful information provided from other related domains [1].

2.3.1 Notations and Definition

Following the notations introduced by PAN and YANG [38], a domain $\mathcal{D}$ consists of two components: a feature space $\mathcal{X}$ and a marginal probability distribution $P(X)$, where $X = \{x_1, \cdots, x_n\} \in \mathcal{X}$. Taking the document classification problem as an example, where each term is a binary feature, $\mathcal{X}$ is the space of all term vectors, x_i is the i^{th} term vector which corresponds to some documents, and X is a particular learning sample. In general, if two domains are different, they may have different feature spaces or different marginal probability distributions.

Given a specific domain $D = \{\mathcal{X}, P(X)\}$, a task consists of two components: a label space $\mathcal{Y}$ and a objective predictive function $f(\cdot)$, and it is denoted by $\mathcal{T} = \{\mathcal{Y}, f(\cdot)\}$. $f(\cdot)$ is a not observed function that can be learned from the training data and can be used to predict unseen examples. Training data consists of pairs $\{x_i, y_i\}$, where $x_i \in X$ and $y_i \in \mathcal{Y}$. We use $f(x)$ to predict the corresponding label of a new instance x. From a probabilistic point of view, $f(x)$ can be written as a conditional probability $P(y|x)$.

Given only one source domain D_S and one target domain D_T, which is the most popular case in literature, we denote the *source domain* as $D_S = \{(x_S, y_S), \cdots, (x_{S_n}, y_{S_n})\}$, where $x_{S_i} \in \mathcal{X}_\mathcal{S}$ is the data instance and $y_{S_i} \in \mathcal{Y}_\mathcal{S}$ is its corresponding label. Similarly, we denote the *target domain* data as $D_T = \{(x_T, y_T), \cdots, (x_{T_n}, y_{T_n})\}$, where $x_{T_i} \in \mathcal{X}_\mathcal{T}$ and $y_{T_i} \in \mathcal{Y}_\mathcal{T}$ is its corresponding label. In most cases, $0 \leq n_t << n_s$. Finally, we can define transfer learning following [38]:

Definition 2.1. ***Transfer Learning:*** *Given a source domain D_S and a learning task T_S, a target domain D_T and a learning task T_T, transfer learning aims to help improve the learning of the target predictive function $f_T(\cdot)$, i.e., the conditional probability distribution $P_T(Y_T|X_T)$, in D_T using knowledge in D_S and T_S, where $D_S \neq D_T$, or $T_S \neq T_T$.*

We illustrate the transfer learning process in Figure 2.3. As can be seen, in a traditional machine learning process (left), models learn from scratch, while transfer learning techniques (right) transfer the knowledge previously learned from one or more solved tasks to a target task that has fewer training data available. It settles the problem of lack of data in the target domain with more knowledge gained from the source tasks.

As a domain is a pair $D = \{\mathcal{X}, P(X)\}$, so the condition $D_S \neq D_T$ implies that either $\mathcal{X}_\mathcal{S} \neq \mathcal{X}_\mathcal{T}$ or $P_S(X) \neq P_T(X)$. Either the term features are different between two sets, or their marginal distributions are different. In the same way, as a task is defined as a pair $\mathcal{T} = \{\mathcal{Y}, P(Y|X)\}$, so the condition $\mathcal{T}_\mathcal{S} \neq \mathcal{T}_\mathcal{T}$ implies that either $\mathcal{Y}_\mathcal{S} \neq \mathcal{Y}_\mathcal{T}$ or $P(Y_S|X_S) \neq P(Y_T|X_T)$. When target and source domains are the same ($\mathcal{D}_\mathcal{S} = \mathcal{D}_\mathcal{T}$) and so are their learning tasks ($\mathcal{T}_\mathcal{S} = \mathcal{T}_\mathcal{T}$), it becomes a traditional ma-

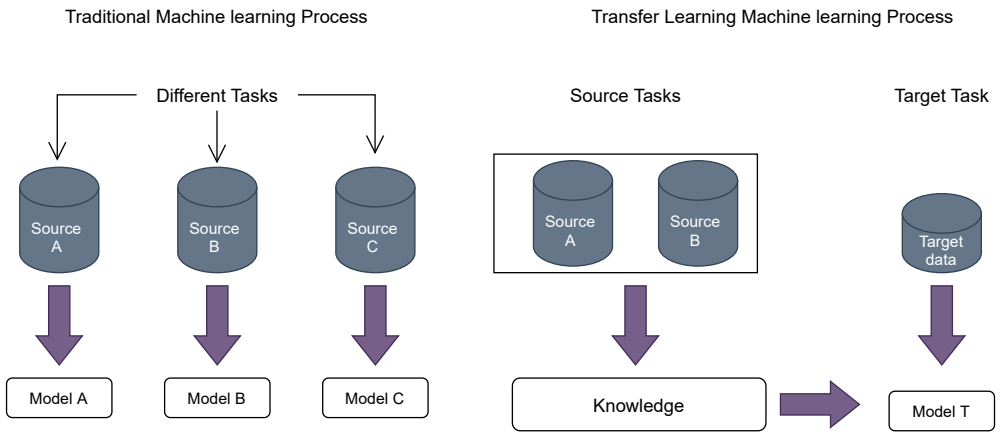

Figure 2.3: Difference between traditional machine learning processes (left) and transfer learning processes (right).

chine learning problem. We have two different scenarios when domains are different: (1) the feature spaces between the domains are different ($\mathcal{X}_\mathcal{S} \neq \mathcal{X}_\mathcal{T}$); (2) the feature spaces between domains are the same but marginal probability distributions between domain data are different ($P(X_S) \neq P(X_T)$, where $X_{S_i} \in \mathcal{X}_\mathcal{S}$ and $X_{T_i} \in \mathcal{X}_\mathcal{T}$). The first scenario corresponds to when the two sets of documents are described in different languages, while the second may correspond to when we have source and target domains focusing on different topics, for example.

Given two specific domains, there are two scenarios for when the learning tasks $\mathcal{T}_\mathcal{S}$ and $\mathcal{T}_\mathcal{T}$ are different: (1) the label spaces between the domains are different ($\mathcal{Y}_\mathcal{S} \neq \mathcal{Y}_\mathcal{T}$); (2) the conditional probability distributions between the domains are different ($P(Y_S|X_S) \neq P(Y_T|X_T)$, where $Y_{S_i} \in \mathcal{Y}_\mathcal{S}$ and $Y_{T_i} \in \mathcal{Y}_\mathcal{T}$). The first scenario is like the source domain has binary document classes, while the target domain has ten classes. The second scenario corresponds to the situation where the source and target documents are very unbalanced in terms of the user-defined classes. If there is an explicit or implicit relationship between the feature spaces of the two domains, we say source and target domains are related.

2.3.2 Transfer Learning Techniques and Approaches

Researches on transfer learning focuses on three issues, which are important to design a transfer learning algorithm [1, 38]:

What to Transfer focuses on answering which part of knowledge can be transferred across domains or tasks. Some knowledge is specific for individual domains or tasks, and some knowledge may be common between different domains such that they may help improve performance for the target domain or task.

When to Transfer asks in which situations transferring is practicable. The goal is to know when knowledge should or should not be transferred. When domains are

not related to each other, brute-force transfer may be unsuccessful, and it can even hurt the performance of learning in the target domain. This is often referred as *negative transfer*. Most of the current studies focus on "what to transfer" and "how to transfer" but how to avoid negative transfer is also an important open issue that is attracting the researcher's attention.

How to Transfer specifies the form that a transfer learning method takes. Different answers to this question give a categorization for transfer learning algorithms [1]:

1. *instance-based* algorithms: knowledge transferred corresponds to the weights attached to source instances;
2. *feature-based* algorithms: knowledge transferred corresponds to the subspace spanned by the features in the source and target domains;
3. *model-based* algorithms: knowledge transferred is embedded in part of the source domain models;
4. *relation-based* algorithms: knowledge to be transferred corresponds to rules specifying the relations between the entities in the source domains.

Last but not least, transfer learning can be categorized into three sub-settings, each based on different situations between the source and target domains and tasks. For more details about transfer learning categorization, please refer to [38].

1. *Inductive transfer learning*: when the target task is different from the source task. In this case, it does not matter if the source and target domains are the same or not. Some labeled data in the target domain are required to *induce* an objective predictive model $f_T(\cdot)$ for use in the target domain. *Inductive transfer learning* can be categorized into two cases according to different situations of labeled and unlabeled data in the source domain:
    1.1 When a lot of labeled data is available, *inductive transfer learning* is similar to the multi-task learning setting. However, *inductive transfer learning* setting only aims at achieving high performance in the target task by transferring knowledge from the source task while multi-task learning tries to learn the target and source tasks simultaneously.
    1.2 There is no labeled data available in the source domain. Thus, *inductive transfer learning* setting is similar to the *self-taught learning* setting [39]. In the latter, label spaces between the source and target domains may be different, which implies that the information of the source domain cannot be used directly.

2. *Transductive transfer learning*: the source and target tasks are the same, but the source and target domains are different. No labeled data in the target domain are available but a lot of labeled data in the source domain are available. It can be categorized into two cases:

   2.1 The feature spaces between the source and target domains are different ($\mathcal{X}_S \neq \mathcal{X}_T$);

   2.2 The feature spaces between domains are the same ($\mathcal{X}_S = \mathcal{X}_T$) but the marginal probability distributions of the input data are different ($P(X_S) \neq P(X_T)$).

3. *Unsupervised transfer learning*: It is similar to *inductive transfer learning* setting, yet, the target task is different from but related to the source task. It focus on solving unsupervised learning tasks in the target domain and there are no labeled data available in both source and target domains in training.

2.3.3 Relation-based Transfer Learning

As for traditional learning methods, relational learning models can also suffer from the lack of data [1]. Besides, changes in the relational domain lead to the learned model performing poorly so it must be rebuild from scratch. For SRL models, the main challenge is how to transfer vocabulary from a source domain into a quite different target domain. Transferring is based on the assumption that the relations among data in the source and target domains have common regularities. There are two mechanisms of relation-based transfer learning:

- *First-order Relation-based*: it assumes that, if two domains are related, they may share some similar relations among data instances, which can be transferred across domains. As an example, the relation *workedunder* between an *actor* and a *director* is analogous to the relation *advisedby* between a *student* and a *professor*.

- *Second-order Relation-based*: it assumes two related domains share some similar relation-independent structural regularities. These regularities can be extracted from the source domain and then transferred to the target domain. Many abstract rules about relations are valid across different real-world domains. As an example, in linguistics, words with similar distributional characteristics tend to be semantically related.

Figure 2.4 shows an example of both mechanisms from a academic domain into the movies domain. Unfortunately, there has not been too much research about

"when to transfer" for relational domains. We describe some of the current work in section 2.5. For more details about relational-based transfer please refer to [1]. In this dissertation, we focus on first-order transfer as we assume predicates are similar given their contexts.

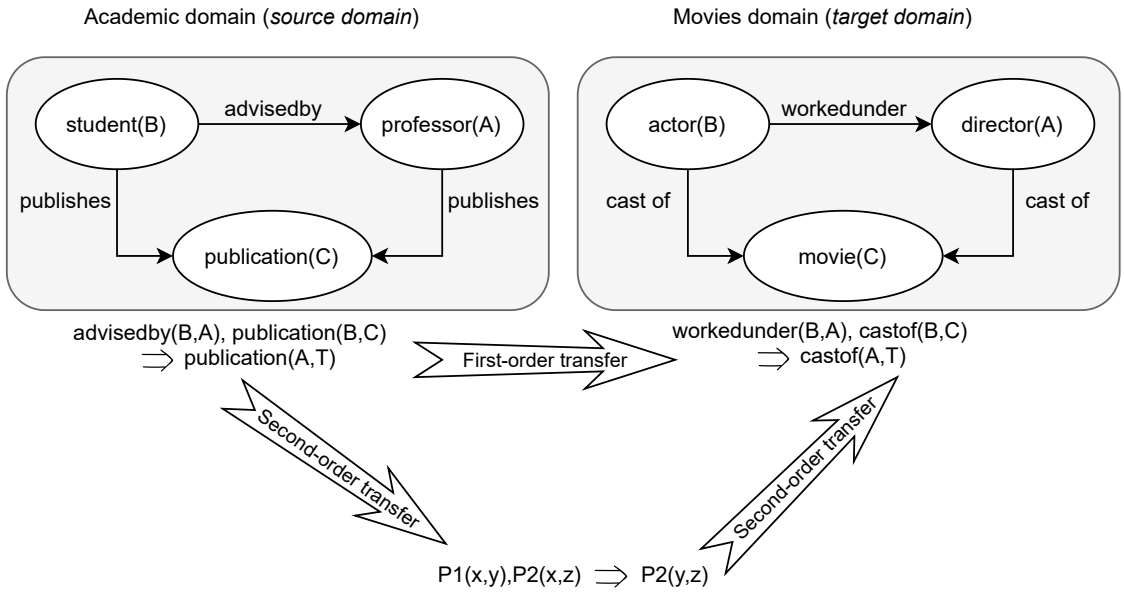

Figure 2.4: Example of relational-based transfer mechanisms [1].

2.4 Words Embeddings

Predicates come in natural language, so we must represent them numerically as we want to compute similarities. For most Natural Language Processing (NLP) tasks, the main challenge is how to represent information expressed in natural language.

For computers, words such as "desk" are nothing but a sequence of characters. Simple approaches like representing words as patterns of bits and one-hot vector representation have some limitations [15]. First, both do not incorporate the semantic information of words because "table" and "desk" are similar words with totally different representations. We want to represent words in a way that encodes the semantics of words. Second, the former is a character-wise representation, so the size of the representation depends on the length of the words (number of characters). A variable size is not desirable to integrate machine learning models and complicates the comparison of representations of different words. The latter grows with the size of the vocabulary, so it is model storage-intensive for large vocabularies.

Proposed by SALTON *et al.* [40], the Vector Space Model (VSM) arises as a solution to the limitations of previous representations. It is the most successful and influential model to encode words, documents, sentences, concepts or entities

as vectors. In this type of representation, objects are represented as vectors in a multi-dimensional continuous space. This space is usually called *semantic space*, and the representation of these objects are called *distributed representations*. The VSM model introduces the notion of similarity as the similarity of two words (vectors) can be measured by their distance in the space. Besides, many more words can fit into a low dimension space. It addresses the storage-intensive issue of one-hot encoding as a vocabulary of size m can fit into an n-dimensional vector space, where $n << m$. All words are placed to the VSM automatically by analyzing word co-occurrences in large text corpora. Word representation learning is usually characterized as an unsupervised or self-supervised learning. Then, there is no need for manual annotation of the training data. This enables the use of raw texts that are available at scale. Word representations generated using neural networks are commonly referred to as *Word Embeddings* [15].

Word Embeddings are the most common and useful way to represent words as dense vectors of fixed length. These vectors have an intuitive interpretation as the meaning of a word is encoded such that words closer in the vector space are expected to be similar in meaning. When it comes to word embeddings, a word is characterized by the company it keeps, so words that appear in similar contexts must have similar meanings [15]. For instance, *student* and *professor* tend to have similar semantics since they usually appear in similar contexts. Word embeddings represent natural language using geometric relations and efficiently transposes discrete word representation into a continuous space, which is why they are widely used in NLP problems [41]. An approach to learning these representations is to train log-bilinear models such as Word2Vec [42] and fast-Text [43]. Both models are based on either Continuous Bag-of-Words (CBOW) or Skip-gram architectures. CBOW architecture predicts missing words using their surrounding context, while Skip-gram aims to predict the words in the surrounding context given a target word. In this work, we focus on the fast-Text representations and the Skip-gram architecture. First, because it covers words that may not appear in the vocabulary; then, we are interested in predicting the context of a given word.

2.4.1 fast-Text

Proposed by BOJANOWSKI *et al.* [43], fast-Text is an extension of the model proposed by MIKOLOV *et al.* [44], in which embeddings are created from sub-words to alleviate issues with out-of-vocabulary words.

Given a word vocabulary of size W, where each word is identified by its index $w \in \{1, \ldots, W\}$, the goal is to learn a vector representation for each word in the vocabulary. Given a large training corpus as a sequence of words $w_1, \ldots, w_T$, the

objective of the Skip-gram model is to maximize the log-likelihood

$$LL = \sum_{t=1}^{T} \sum_{c \in C_t} \log p(w_c | w_t), \tag{2.5}$$

where C_t is the context, the set of indices of words surrounding word w_t. Then, the probability of observing a context word w_c given w_t will be parameterized using the mentioned word vectors. Suppose we have a scoring function s that maps pairs of *(word, context)* to scores in $\mathbb{R}$. The problem of predicting context words is modeled as a set of independent binary classification tasks, where the goal is to predict the presence or absence of context words. Given a word at position t, called w_t, all context words are considered as positive examples and negative samples are chosen at random from a dictionary. For a context position c, using binary logistic loss, we obtain the negative log-likelihood

$$NLL = \log \left(1 + e^{-s(w_t, w_c)}\right) + \sum_{n \in \mathcal{N}_{t,c}} \log \left(1 + e^{s(w_t, n)}\right), \tag{2.6}$$

where $\mathcal{N}_{t,c}$ is a set of negative examples chosen at random from the vocabulary. Denoting the logistic loss function $\ell : x \longmapsto log(1 + e^{-x})$, the objective function can be re-written as

$$\sum_{t=1}^{T} \Big[\sum_{c \in C_t} \ell(s(w_t, w_c)) + \sum_{n \in \mathcal{N}_{t,c}} \ell(-s(w_t, n)) \Big]. \tag{2.7}$$

The scoring function s can be defined as the scalar product between word and context vectors. Suppose a word w_t and a context word w_c, we can define two vectors u_w and v_w in $\mathbb{R}^d$ for each word in the vocabulary. These are commonly referred as *input* and *output* vectors. Then, the score can be computed as $s(w_t, w_c) = \mathbf{u}_{w_t}^T \mathbf{v}_{w_c}$, where vectors $\mathbf{u}_{w_t}$ and $\mathbf{v}_{w_c}$ correspond to words w_t and w_c, respectively.

The model described above is the Skip-gram model with negative sampling, proposed by MIKOLOV *et al.* [44]. This model ignores the internal structure of words by using a distinct vector representation for each word. To incorporate this information, BOJANOWSKI *et al.* [43] proposed a different scoring function s. In the fast-Text model, each word w is represented as a bag of character n-gram. Special boundary symbols $<$ and $>$ are added to the beginning and the end of words, allowing to distinguish prefixes and suffixes from other character sequences. The word w is also included in the set of its n-grams, to learn a representation for each word in addition to character n-grams. As an example, taking the word *where* and $n = 3$, it will be represented by the character n-grams:

$$< wh, whe, her, ere, re >$$

and the special sequence

$$< where > .$$

It is important to say that the sequence $< her >$, which correspond to the word HER, is different from the tri-gram *her* from the word *where*. Then, given a dictionary of n-grams of size G and a word w, the set of n-grams appearing in w can be denoted by $\mathcal{G}_w \subset \{1, \ldots, G\}$. A vector representation $\mathbf{z}_g$ is associated to each n-gram g and a word is represented by the sum of the vector representations of its n-grams, which leads to the scoring function

$$s(w, c) = \sum_{g \in \mathcal{G}_w} \mathbf{z}_g^T \mathbf{v}_c. \tag{2.8}$$

2.4.2 Similarity Metrics

Word vectors afford useful operations like addition, subtraction, distance measures, among others [41]. In this work, we apply three distance measures to find a suitable mapping between a pair of predicates:

Euclidean Distance is the simplest way to measure distance between two real-valued vectors. The distance between two vectors $\vec{p}, \vec{q} \in \mathbb{R}^n$ is given by:.

$$d(p, q) = ||p - q||_2. \tag{2.9}$$

Soft Cosine Measure is a modification of the traditional cosine similarity measure as it takes into account the similarity between features (words) in the VSM [45]. It considers the cosine similarity of each pair of features to build a matrix of similarity s which introduces new features to the VSM. Thus, the Soft Cosine similarity between two vectors $\vec{p}, \vec{q} \in \mathbb{R}^n$ is given by Equation 2.10. If there is no similarity between features, $s_{ii} = 1$ and $s_{ij} = 0$ for $i \neq j$, is equivalent to the traditional cosine similarity measure.

$$soft\_cosine(p, q) = \frac{\sum \sum_{i,j}^{N} s_{i,j} p_i q_j}{\sqrt{\sum \sum_{i,j}^{N} s_{i,j} p_i p_j} \sqrt{\sum \sum_{i,j}^{N} s_{i,j} q_i q_j}} \tag{2.10}$$

Word Mover's Distance (WMD) also considers the semantic similarity between word pairs [46]. It is a special case of Earth Mover's Distance [47] as it considers the "travel cost" between words to obtain the minimum cumulative cost of moving a given document $\mathbf{d}$ to a document $\mathbf{d}'$. It takes an embedding matrix $\mathbf{X} \in \mathbb{R}^{d \times n}$ for a finite size vocabulary of n words, where the i^{th} column, $\mathbf{x}_i \in \mathcal{R}^d$, represents the embedding of the i^{th} word in d-dimensional space. It assumes text documents represented as normalized bag-of-words (nBOW) vectors, $\mathbf{d} \in \mathcal{R}^n$, where

the vector $\mathbf{d}$ is a point on the $n-1$ dimensional simplex of word distributions. If word i appears c_i times in the document, then $d_i = \frac{c_i}{\sum_{j=1}^{n} c_j}$. Thus, documents with no words in common will lie in different regions of the simplex, but may still be semantically close. In the end, the nBOW vector $\mathbf{d}$ becomes very sparse as most words will not appear in every document.

The distance between words i and j is obtained by the Euclidean distance, $c(i,j) = ||\mathbf{x}_i - \mathbf{x}_j||_2$, which is the cost associated with "traveling" from one word to another. Thereby, semantic similarity between feature pairs is incorporated to the distance metric. Finally, WMD solves the linear transportation program presented in 2.11 to obtain how costly it is to travel from one document to another.

$$\begin{aligned} \min_{\mathbf{T} \geq 0} \quad & \sum_{i,j=1}^{n} \mathbf{T}_{ij} c(i,j) \\ \text{subject to:} \quad & \sum_{j=1}^{n} \mathbf{T}_{ij} = d_i \; \forall_i \in \{1,...,n\} \\ & \sum_{i=1}^{n} \mathbf{T}_{ij} = d'_j \; \forall_j \in \{1,...,n\}. \end{aligned} \tag{2.11}$$

As $\mathbf{T} \in \mathcal{R}^{n \times n}$ is a sparse flow matrix where $\mathbf{T}_{i,j} \geq 0$ denotes how much of word i in $\mathbf{d}$ travels to word j in $\mathbf{d}'$. To transform $\mathbf{d}$ into **d'**, the entire outgoing flow from word i must be equal d_i, as guaranteed by constraint $\sum_j \mathbf{T}_{ij} = d_i$. Similarly, the incoming flow to word j must be equal d'_j, as guaranteed by constraint $\sum_i \mathbf{T}_{ij} = d'_j$. Last but not least, the objective function is the minimum (weighted) cumulative cost required to move all words from $\mathbf{d}$ to **d'**.

Relaxed Word Mover's Distance (RWMD) is a variation of WMD which optimizes computation by relaxing the transportation problem [46]. For datasets with many unique words or a large number of documents, solving WMD optimal transport is nonviable, as the best average time complexity for WMD scales $O(p^3 log p)$, for p denoting the number of unique words in the documents [48]. In addition, much tighter bounds are obtained by relaxing WMD. Relaxation is done by removing one of the two constraints. Removing both constraints results in the trivial lower bound $\mathbf{T} = 0$. By removing the second constraint, the relaxed linear transportation program becomes

$$\begin{aligned} \min_{\mathbf{T} \geq 0} \quad & \sum_{i,j=1}^{n} \mathbf{T}_{ij} c(i,j) \\ \text{subject to:} \quad & \sum_{j=1}^{n} \mathbf{T}_{ij} = d_i \; \forall_i \in \{1,...,n\}, \end{aligned} \tag{2.12}$$

which yields a lower-bound to the WMD distance since every WMD solution

(satisfying both constraints) is still a feasible solution if one constraint is removed.

2.5 Related Work

Graph Neural Networks (GNNs) have widely been used to learn structured data, and some of the work includes the use of Transfer Learning techniques [49–51]. However, in this work, we only focus on symbolic representation. In the literature, there are works focused on either transferring similar relations or structural regularities. Nevertheless, the main question that remains is how to transfer vocabulary considering different domains.

LTL [12] focus on finding structural regularities as it performs type-matching to identify predicates in the target domain that are similar in their relational structure to predicates in the source domain. It performs type-based tree construction as it tries to match paths between the source and target tree structures. Paths in the source domain match paths in the target domain if the same number of arguments are related to each link path. After transferring trees, it uses theory refinement in each clause by adding or deleting predicates to try to improve its accuracy.

The TAMAR [13] algorithm assumes domains share similar relations. It uses weighted pseudo-log-likelihood (WPLL) to transfer MLNs from a source domain to a target domain. It performs an exhaustive search through the space of all legal mappings. A mapping is legal if each source predicate in a given clause is mapped to a compatible target predicate or to "empty". Predicates are compatible if they have the same arity and if the types of arguments match the current type constraints. The mapping that gives the best WPLL in the target domain is used as mapping for a clause in the source domain. In the end, it revises the structure using a algorithm similar to FORTE [52] to improve its accuracy.

An extension of TAMAR, SR2LR [53] also considers the transfer of MLNs by producing type-consistent mappings. The main difference is that SR2LR deals with minimal target data as it considers only one entity is available (*single-entity-centered* case) in the target domain. Then, the theory is generalized for more than one entity.

GROOT [54] differs from previous approaches as it relies on a genetic algorithm to find mappings for transferring RDNs. Each individual is composed of chromosomes which are feasible mappings and a fitness function value. Chromosomes correspond to trees. An individual is a set of trees and each node of its trees is an alelo. Mapping is performed randomly but restricted to the arity of predicates and type-consistence. Predicates are mapped sequentially and respecting the order they appear in the trees.

TODTLER [55] tries to generalize regularities from one model to another based on second-order clauses. Thus, knowledge transfer is viewed as the process of learn-

ing a declarative bias in one domain. It proposes a two-stage procedure that learns which second-order patterns are useful in the source domain to bias the learning process in the target domain towards models that have the same patterns.

TreeBoostler [3] is the most closely related to our algorithm as it also focuses on transferring RDNs. It follows the same concept of legal mappings defined by MIHALKOVA *et al.* [13]. A mapping is legal if predicates share the same arity and the types of their arguments agree. It performs an exhaustive search by building type constraints to find adequate mappings for a source predicate in the target domain. Weighted variance is used as the decision criterion to choose the best adequate mapping given the structure of the source regression trees. Then, to improve its accuracy, it revises those trees by pruning and expanding nodes. Our algorithm proposes a modification to TreeBoostler's mapping component. It takes advantage of the semantics of pre-trained word vectors to find mappings by similarity. In this way, there is no need for searching the whole space of possible mappings. In the end, it refines the clauses by pruning and expanding trees to better fit the target domain, following the same approach as proposed by AZEVEDO SANTOS *et al.* [3].

There are many applications of embeddings for relational tasks such as TransE [56] that represents relationships as translations in the embedding space. Given two entities h and t, and a relation l, it follows an energy-based framework to state that both are related if $h + l \approx t$, which means t should be the nearest neighbor of $h + l$. Based on TransE, TransH [57] proposes improvements to TransE flaws by enabling an entity to have distributed representations when it is involved in different relations. Relations are translating operations on a hyperplane and entities can have multiple representations accordingly to a relation. However, entities may be similar and close to each other but may be different in some specific aspects and far from each other in the corresponding relation spaces. Thus, TransR [58] models entities and relations in distinct spaces, entity and multiple relation spaces, and learns embedding via translation between projected entities. In [59], authors investigate neural-net embeddings to assist the classification of relational data instances and show evidence that embedded representations can be useful for problems poor in domain knowledge, but results depend on the embedding method used.

Chapter 3

TransBoostler: word embeddings-based transfer learning algorithm

This chapter introduces our proposed algorithm, a word embeddings-based transfer learning approach for relational domains named TransBoostler. It comprises the same top-level components as proposed by AZEVEDO SANTOS *et al.* [3]. First, the source boosted trees structure is transferred to a different target domain by finding adequate predicate mappings by similarity. Next, the algorithm revises the mapped trees by pruning and expanding nodes to fit the target data better. The most significant difference compared to TreeBoostler is how our algorithm finds adequate predicate mappings. It tries to solve the problem of transferring vocabulary by mapping predicates by similarity.

3.1 Transferring the Structure

Following TreeBoostler [3], TransBoostler also adopts the local mapping approach introduced by MIHALKOVA *et al.* [13]. It consists of mapping predicates as they appear in each source clause, so mapping is performed separately and independently of how other clauses were mapped. The other approach, called global mapping, differs from the local approach as it establishes a mapping for each source predicate to a target predicate, and this mapping is used to translate the entire source trees. The global mapping is computationally costly as the size of the search space grows exponentially with the number of predicates in the source domain. Local mapping is generally more scalable since the number of predicates in a single clause is smaller than the total number of predicates of the source domain.

Each path from the root to the leaf in a relational regression tree can be seen

as a clause in a logic program. RDN-Boost works on a set of relational trees, so these paths are not independent of each other as they may share the same inner nodes. Then, once a predicate in the source domain is mapped, its corresponding target mapping is propagated to the entire structure. In this way, every inner node is translated according to the translations already found, even when using the local mapping. Transference starts from the root node of the first source tree and works recursively to find the best mapping for non-mapped predicates. Algorithm 3.1 presents the top-level transfer algorithm described. Given the theory learned in the source domain and adequate mappings, our proposed algorithm replaces every source predicate with its corresponding mapping. Figure 3.1 presents an example of RRT for transferring from IMDB to UW-CSE. The resulting structure (right) is the starting point, so the algorithm continues learning to fit the target data. We detail the mapping component in the next section.

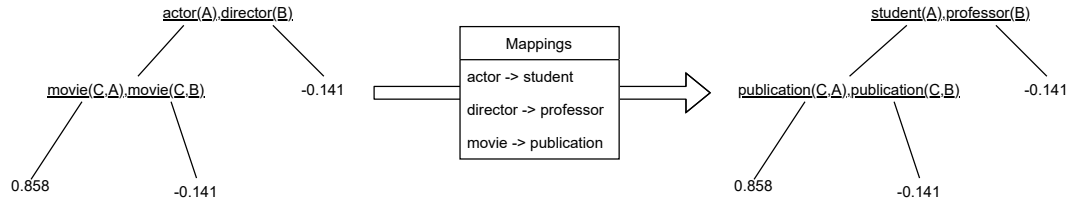

Figure 3.1: Example of transference from IMDB (left) to UW-CSE (right) given the source structure and the corresponding mappings.

3.2 Mapping Component

One of the differences between SRL models and traditional machine learning models is the richness of their vocabulary. Nevertheless, we focus on only transferring predicates. Assuming domains can be related, the main challenge is how to find relations between predicates from a source to a target domain. We also assume predicates have meaningful names, so TransBoostler takes advantage of the semantic of pre-trained word vectors to find a suitable mapping by similarity. To do it, TransBoostler first builds a list of pairs of predicates ordered by similarity, computed with similarity metrics over the embeddings of the predicates. Next, it can map predicates as they appear in structure, so predicates closer to the root have priority, or it follows an ordered list of pairs by similarity to employ the most similar mappings.

3.2.1 Text Normalization

Preprocessing is an important step in many NLP tasks since textual data need to be standardized. In relational datasets, predicates can be made of one or more words.

Algorithm 3.1: Top-Level TransBoostler Transfer Algorithm

```
Input: theory, a set of regression trees
Input: tarPreds, a set of predicates from target domain
Function TRANSFER(theory):
    transferred ← 0
    for tree ∈ theory do
        new_tree ← 0
        TRANSFER_TREE(node.root, new_tree)
        Append new_tree to transferred
    end
return transferred
Function TRANSFER_TREE(node, transfer_node):
    if node is leaf then
        Define transfer_node as leaf
        Stop procedure
    end
    predicates ← Get set of predicates not mapped from node
    if predicates is empty then
        new_node ← Translates predicates in node
        transfer_node ← new_node
    end
    else
        corresponding_mappings ← given predicates get the most similar
          predicates
        Update the global variable mappings
    end
    if transfer_node is not empty then
        Call TRANSFER_TREE(node.left, transfer_node.left)
        Call TRANSFER_TREE(node.right, transfer_node.right)
    end
    else
        if node.left is leaf then
            Call TRANSFER_TREE(node.right, transfer_node)
        end
        else
            Append node.right to the right-most path of node.left
            Call TRANSFER_TREE(node.left, transfer_node)
        end
    end
return transferred
```

As we want to turn words into vectors using a pre-trained model, the first step of the mapping process is to split each predicate into its component words. In this way, we may ensure to get the corresponding vectors to each word.

To perform word segmentation, we use Ekphrasis [60] with Wikipedia corpora. Ekphrasis is a text processing tool which performs many preprocessing tasks, including word segmentation, which is used for segmenting hashtags. Word segmentation is done using the Viterbi algorithm [61] and uses word statistics of unigrams and bigrams from unlabeled data to obtain word probabilities. In this way, predicates such as *athleteplaysforteam* can be segmented into its component words *athlete plays for team*. Furthermore, some predicates or words can appear in their shortened form, which is the case of "*ta*" that stands for *teaching assistant* and "*tempadvisedby*" that stands for *temporarily advised by*. After word segmentation, every shortened word is replaced by its full form using a pre-built dictionary. Thus, "*ta*" becomes "*teaching assistant*" and "*tempadvisedby*" becomes "*temporarily advised by*".

Lastly, we can have inflected forms like *members* and *member*. These words have the same *lemma*. We call a lemma a set of lexical forms having the same stem, the same major part-of-speech, and the same word sense [62]. To handle plural nouns and also turn verbs into their base form, we use WordNet lemmatizer [63]. By applying lemmatization, we can determine if words have the same root besides surface differences. Examples are words *sang*, *sung*, and *sings*, which are forms of the verb *sing* [62]. Another example, words *am*, *are*, and *is* share the same lemma *be*. The lemmatizer maps all inflected forms to their base form. Verbs and plural nouns are identified in predicates by a Part-Of-Speech Tagger (POS Tagger) tool [64]. Given a sequence of tokenized words like *Janet will back the bill* and a tagset, the output is a set of tags that correspond to each word. In this case, *noun, aux, verb, det, noun*. As words are ambiguous, *book* can be a verb or a noun, the goal is to find the correct tag for the situation by resolving ambiguities and choosing the proper tag for the context [62]. If a word is tagged as a verb or a plural noun, it is turned into its corresponding base form.

3.2.2 Word-Vectors Representation

As we want to compute similarities, we must represent each predicate numerically. Then, word vectors pre-trained on Wikipedia with fast-Text Skip-gram [65] are used to represent each predicate into the VSM. The use of pre-trained word vectors is essential here as the predicates of relational datasets constitute a very limited vocabulary. Also, pre-trained word vectors contribute to finding more similar predicates as similar words are approximated by context. If a word does not belong to the pre-trained model vocabulary, it is represented as a null vector.

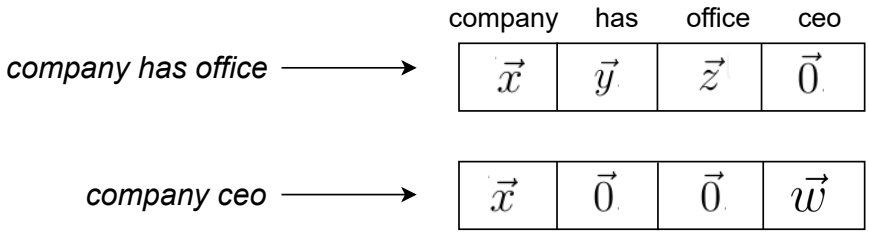

Figure 3.2: Example of representation such that "company has office" and "company ceo" are in the same feature space.

Representing each component word of a predicate as its corresponding word-vector might result in a set of vectors. When applying Euclidean distance, word vectors in the same predicate are concatenated to become one single vector. Concatenation avoids information loss since we have very short sentences. In the example of Figure 3.2, we would like to compute the similarity between predicates *company has office* and *company ceo*. However, *company has office* is a 3-dimensional vector while *company ceo* is 2-dimensional. The former has components $[\vec{x}, \vec{y}, \vec{z}]$ and the latter $[\vec{x}, \vec{w}]$. We must express both in the same feature space before concatenating. To do it, two new vectors whose components represent terms in predicates are created. Then, we have m-dimensional vectors, where m is the number of unique words in both predicates. In our example, it results in two new 4-dimensional vectors. The word "company" is the only one they have in common, so the word vector $\vec{x}$ appears in the same position in both arrays. Words that do not appear in a predicate are represented as a null vector of the same dimension. To express *company has office*, we set its components to $[\vec{x}, \vec{y}, \vec{z}, \vec{0}]$. To express *company ceo*, we set its components to $[\vec{x}, \vec{0}, \vec{0}, \vec{w}]$.

3.2.3 Mapping by Similarity

A source predicate is mapped to a target predicate if they have the highest similarity value in comparison with other target predicates. Suppose a source predicate *movie* and a list of predicates of the target domain [*author*, *sameauthor*, *venue*]. Then, *movie* is mapped to *author* if and only if $sim(movie, author) > sim(movie, sameauthor)$ and $sim(movie, author) > sim(movie, venue)$, where sim is the similarity function. To maintain variables' consistency, we only consider predicates of the same arity. If there is a tie, we follow alphabetical order. A predicate is considered as an adequate mapping if it follows Definition 3.1. The comparison between predicates stands for alphabetical order. It is not permitted to have more than one distinct source predicate mapped to the same target predicate. If the algorithm cannot find a compatible mapping, the predicate is mapped to "empty".

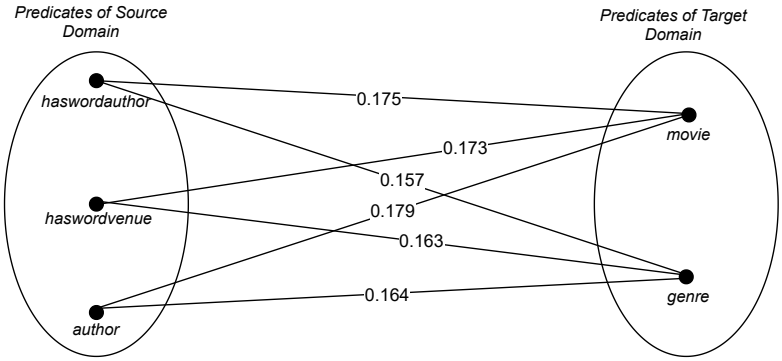

Figure 3.3: Representation of the ranked-first mapping as the Maximum-Weight Bipartite Matching problem when transferring from Cora to IMDB.

Definition 3.1. *Let $p(X_1, \cdots, X_n)$ be an atom in the source vocabulary (D_S) consisting of a predicate p and arity n. Let $q(Z_1, \cdots, Z_m)$ be an atom in the target vocabulary (D_T) consisting of a predicate q and arity m. We say that q/m is the corresponding mapping of p/n if $m = n$ (they have the same arity), and if $sim(p, q) \geq sim(p, i)$ and $q > i$, $\forall i \in \{D_T - q\}$.*

When mapping predicates to their most similar, the order we perform mapping matters. We can either follow the order that predicates appear in the source structure or follow the rank of similarities. We propose two approaches to perform mapping by similarity: (1) it follows the order in which predicates appear in the source structure, so predicates that appear closer to the root have priority to be mapped to their similar corresponding targets, or; (2) it ignores the order in which predicates appear in nodes of trees and follows unrestrictedly the ordered list of similarities between pairs of source and target predicates. This approach prioritizes the pairs at the top of the rank. It is a greedy solution to the problem of finding maximum-weight matchings in a bipartite graph, where the two disjoint sets U and V of vertices are sets of predicates from the source and target domains, respectively. Then, we have a bipartite graph with weighted edges to connect a vertice from U to V. Weights are given by the similarity between predicates. Figure 3.3 presents an example based on Table 3.1. We call the former approach *depth-first mapping* and the latter *ranked-first mapping*. The first approach is presented in Algorithm 3.2 and the second in Algorithm 3.3.

Suppose we want to map predicates from Cora to IMDB. Suppose the theory has three predicates in its structure which appear in the following order: *haswordauthor*, *author*, and *haswordvenue*. After building pairs of predicates, similarities are computed using WMD and presented in Table 3.1. The corresponding mappings are presented in Table 3.2 for *depth-first mapping* and *ranked-first mapping*. When performing the *depth-first* mapping, the predicates *haswordauthor* and *author* are mapped to *movie* and *genre*, respectively, in the target domain. As source

Table 3.1: Similarities between pairs of predicates of the same arity in Cora and IMDB domains using WMD.

	Similarity
haswordauthor, movie	0.175
haswordauthor, genre	0.157
haswordvenue, movie	0.173
haswordvenue, genre	0.163
author, movie	0.179
author, genre	0.164

Table 3.2: Mappings when using depth-first (left) and ranked-first (right) mapping approaches for transferring Cora → IMDB based on Table 3.1.

Depth-First		Ranked-First	
haswordauthor(A,B)	→ movie(A,B)	author(C,A)	→ movie(C,A)
author(C,A)	→ genre(C,A)	haswordvenue(D,B)	→ genre(D,B)
haswordvenue(D,B)	→ *empty*	haswordauthor(A,B)	→ *empty*

predicates cannot be mapped to the same targets, there are no predicates left for *haswordvenue* because IMDB has only two predicates of arity two. When performing *ranked-first* mapping, it is the predicate *haswordauthor* that is mapped to empty since the predicates *haswordvenue* and *author* are closer to the predicates *genre* and *movie*, respectively.

After transfer, there are three different scenarios as modeled by AZEVEDO SANTOS *et al.* [3]: (1) the best scenario is when all literals in an inner node have a non-empty predicate mapping, which means we were able to find a mapping for each predicate in the source trees; (2) an inner node has one or more predicates mapped to "empty", but at least one is mapped to non-empty. In this case, predicates mapped to empty are discarded; (3) an inner node has all its literals mapped to an empty predicate. This is the worst case because discarding all literals results in an empty node, which affects the tree structure. Then, 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, the right child is promoted. If both are leaves, it is discarded.

3.3 Theory Revision

Mapping only vocabulary is usually not enough as knowledge comes from a different distribution domain [13]. To repair possible faults that can prevent theories from predicting examples correctly, our proposed algorithm also uses Theory Revision [16]. The theory revision component searches for points in theory that are

Algorithm 3.2: Depth-first mapping by similarity given (ordered) source and target lists of predicates.

```
Function MAP_PREDICATE(srcPreds, tarPreds):
    similarities ← {}
    mappings ← {}
    for srcPred ∈ srcPreds do
        srcPredWordVector ← PRE_PROCESS(srcPred)
        for tarPred ∈ tarPreds do
            if sameArity(srcPred, tarPred) then
                tarPredWordVector ← PRE_PROCESS(tarPred)
                sim ← Compute similarity between srcPredWordVector and
                  tarPredWordVector word vectors
                Insert sim to similarities
            end
        end
        Sort similarities
        mostSimilar ← get the most similar target predicate using
          similarities
        Insert mostSimilar to mappings
    end
    return mappings
return mappings
```

Algorithm 3.3: Ranked-first mapping by similarity given source and target lists of predicates.

```
Function MAP_PREDICATE(srcPreds, tarPreds):
    similarities ← {}
    for srcPred ∈ srcPreds do
        srcPredWordVector ← PRE_PROCESS(srcPred)
        for tarPred ∈ tarPreds do
            if sameArity(srcPred, tarPred) then
                tarPredWordVector ← PRE_PROCESS(tarPred)
                sim ← Compute similarity between srcPredWordVector and
                  tarPredWordVector word vectors
                Insert sim to similarities
            end
        end
    end
    Sort similarities in ascending order
    mappings ← get the most similar predicate using similarities
    return mappings
return mappings
```

responsible for misclassified examples, and it proposes modifications to adjust the initial mapped source theory to fit the target data, hence, improving its inferential capabilities. We follow the same approach as proposed by AZEVEDO SANTOS *et al.* [3] which is briefly described in the following.

Every path in trees that is responsible for the misclassification of examples is defined as a *revision point*, i.e., its weighted variance is greater than a given threshold δ. If positive examples are not covered (i.e. false negatives) by the theory, it means the theory is too specific and needs to be generalized, so we call it a *specialization point*. If theory covers negative examples, it is too general and needs to be specialized, so we call it a *generalization point*. These points must be modified to increase accuracy. Modifications are proposed to revision points by applying *revision operators*. First, it applies the *pruning operator* (generalization operator) to increase the coverage of examples by deleting nodes from a tree. It prunes the tree from the bottom to the top recursively, removing nodes whose children are leaves marked as revision points. Secondly, it applies the *expansion operator* (specialization operator). It decreases the coverage of examples by expanding nodes in each tree as it recursively adds nodes that give the best split in a leaf marked as a revision point.

The pruning procedure could prune an entire tree. If this happens, the revision algorithm would have to expand nodes from an empty tree, which is the same scenario as learning from scratch. If pruning results in a null model, deletion of all trees, the operator is ignored as if it was never applied. In the end, both transferred and revised theory are scored using the conditional log-likelihood. If the revised theory scores better than before, it is implemented. To provide the reader with a general understanding of the revision component, Algorithm 3.4 presents the revising procedure along with the revision operators.

Algorithm 3.4: Top-Level Theory Revision Algorithm [3]

Input: *theory*, a set of regression trees
Function `REVISION(theory)`:
    $newTheory \leftarrow \{\}$
    for $tree \in theory$ **do**
        $newTree \leftarrow$ PRUNING($tree$)
        Append $newTree$ to $newTheory$
    end
    if $newTheory$ *is null* **then**
        $newTheory \leftarrow theory$
    end
    for $tree \in newTheory$ **do**
        $tree \leftarrow$ EXPAND_NODES(tree)
    end
    Compute score $theory$ and $newTheory$
    if $score_{newTheory} > score_{theory}$ **then**
        return $newTheory$
    end
    else
        return $theory$
    end
Function `PRUNNING(node)`:
    $left \leftarrow$ PRUNNING($node.left$)
    $right \leftarrow$ PRUNNING($node.right$)
    if $left$ *is leaf and* $right$ *is leaf* **then**
        if $left.variance > \delta$ *and* $right.variance > \delta$ **then**
            Remove node from $node$ and put a leaf in its place
        end
    end
    return node
return node
Function `EXPAND_NODES(node)`:
    $left \leftarrow node.left$
    if $left$ *is a leaf and* $left.variance > \delta$ **then**
        $bestNode \leftarrow$ Find new node that gives the best split
        Add $bestNode$ to $left$
        $left \leftarrow$ EXPAND_NODE($left$)
    end
    $right \leftarrow node.right$
    if $right$ *is a leaf and* $right.variance > \delta$ **then**
        $bestNode \leftarrow$ Find new node that gives the best split
        Add $bestNode$ to $right$
        $right \leftarrow$ EXPAND_NODE($right$)
    end
    return node
return transferred

Chapter 4

Experiments and Results

In this chapter, we present the experiments performed to evaluate the proposed algorithm and the results obtained from them. To investigate the research questions presented in this dissertation, we have performed two experiments. The first one is a simulation of a transfer learning environment with few data available; the second simulates a scenario with increasing amounts of target data.

4.1 Research questions

We conducted a set of experiments for each mapping approach to investigate the following research questions regarding the similarity-based transfer learning approach and baselines. We present the results for depth-first and ranked-first mappings separately.

Q1 Does TransBoostler learn more accurate models than the baselines?

Q2 Can TransBoostler transfer theories by relying on word embeddings similarity?

Q3 How important is revising the theory when transferring relies on word embeddings-based similarity?

Q4 Is the mapping by similarity approach faster than the baselines?

Q5 Does TransBoostler perform better than the baselines with increasing amount of examples in the target data?

The question **Q1** addresses if the proposed method improves learning in relational domains. Also, it concludes if it outperforms learning from scratch and a previous transfer learning approach. Question **Q2** is important to address if transferring the vocabulary by relying on word embeddings-based similarity is cogent. We would like to know if TransBoostler transfers well across domains by assuming

predicates are related by their contexts. Also, we investigate how it depends on theory revision to improve theories inference capacities (**Q3**). Question **Q4** is a common question in the machine learning environment as one of the objectives of applying transfer learning is to reduce training time. In addition, it is desirable to build an approach that is less time-consuming than previous works. Question **Q5** evaluates the proposed algorithm for different numbers of examples to evaluate if TransBoostler tends to make learning a new task less data-consuming.

4.2 Datasets

We evaluated TransBoostler using six publicly available relational datasets paired as in previous literature [3, 12, 13, 53]:

IMDB dataset [66] contains information such as director, actor, genre, and movie. The goal is to use the relations between the entities to predict which actor has worked for a director by learning the *workedunder* relation. It is divided into five mega-examples where each one presents information about four movies. A mega-example is a large set of connected facts [67]. Mega-examples are disconnected and independent of each other.

Cora [68] is a dataset of Computer Science research papers. It contains 1295 distinct citations to 122 papers which were segmented into fields like author, title, venue. Entities can also be connected by relations *sametitle*, *samebib*, and *sameauthor* to indicate if two papers have the same title, the same bibliography, or the same author, respectively. There are also relations to consider words in entities' names like *haswordauthor*, *haswordtitle*, and *haswordvenue*. It is divided into five mega-examples and its goal is to predict if two venues represent the same conference by learning the *samevenue* relation.

UW-CSE [69] contains information about the Department of Computer Science and Engineering at the University of Washington (UW-CSE) represented by publications and their authors, projects and their members, courses levels, etc. It consists of five mega-examples to predict the *advisedby* relation (a student advised by a professor).

Yeast protein [18] is a dataset obtained from MIPS[1] Comprehensive Yeast Genome Database and contains information about proteins with their location, function, enzyme, complex, and phenotype. The goal is to predict if a protein is associated with a class. It consists of four folds independent of each other.

Twitter [55] dataset consists of tweets about Belgian soccer matches divided into two independent folds. It contains words tweeted, relations between accounts

[1]Munich Information Center of Protein Sequence

Table 4.1: Statistics of the six datasets used to evaluate TransBoostler.

Dataset	Number of Constants	Number of Types	Number of Predicates	Number of Positive Examples	Total number of ground literals
IMDB	297	3	6	382	71824
UW-CSE	914	9	14	113	16900
Cora	2457	5	10	3017	152100
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

(following) and the type of accounts which can be a club, fan, or news. The goal is to predict the account type.

NELL [70] is a machine learning system that extracts information from web texts and converts it into a probabilistic knowledge base. We consider two domains from NELL: Sports and Finances. The former contains information about the athlete and their teams, leagues and their teams, etc. The goal is to predict which sport is played by a team. The latter contains information about economic sectors of companies, companies' CEOs, companies' country, etc. The goal is to predict if a company belongs to an economic sector. As in [3], we split the target data randomly into three different folds.

Statistics about all datasets are presented in Table 4.1. The total number of ground literals is the number of all true ground literals consisting of grounding the predicates with constants of their respective types.

4.3 Experimental Methodology

In this section, we present the experiments performed to evaluate TransBoostler. We investigate the research questions for the two mapping approaches proposed in this dissertation.

We follow the same experimental setup as TreeBoostler, so we set the depth limit of trees to be 3, the number of leaves to be 8, the number of regression trees to 10, the maximum number of literals per node to 2, subsampling of negative examples is in a ratio of 2 negatives for 1 positive, and the initial potential is -1.8. We test our algorithm with all the negative examples. TransBoostler is evaluated with four similarity metrics: Soft Cosine, Euclidean distance, WMD, and RWMD. The pre-trained fast-text model used in this work represents words as vectors of 300 dimensions.

To evaluate how revising can improve transferring, we consider two versions

of our algorithm: one only considers predicate mapping and parameter learning (TransBoostler*) and the completed version that additionally performs theory revision (TransBoostler). We also compare results with the same two versions of TreeBoostler. We used conditional log-likelihood (CLL), area under the ROC curve (AUC ROC), area under the PR curve (AUC PR) [71] and training time as measures to compare performance. We did not consider the time required to load the fast-Text model, as it is negligible, but we did consider time to calculate similarities between predicates.

The first experiment simulates the transfer learning scenario by learning from a reduced set of data. Following the previous literature, training is performed on one fold and testing on the remaining $n - 1$ folds. Results are averaged over n runs, where, for each run, a new learned source model is used for transference. We measured the statistical significance between TransBoostler and the baselines using a paired t-test with $p \leq 0.05$. For the second experiment, we compare the performance of TransBoostler for different amounts of target data. We employed traditional cross-validation methodology as training is performed on $n - 1$ folds and testing on the remaining one. Training data is shuffled and divided into five sequence parts. As in the previous experiment, the process is done in n runs, and curves obtained by averaging the results.

We compare our results with RDN-Boost [2], when learning from the target dataset from scratch, and TreeBoostler [3], which is a SRL transfer learning approach. We used the Google Cloud platform for all experiments. Experiments were conducted on an N2 virtual machine with Debian 10, 8 vCPUs, and 32GB of RAM.

4.4 Results

In this section, we present our experimental results. We call RDN-Boost as RDN-B in tables for shorthand. As TreeBoostler was successfully compared with other transfer methods, these results are omitted.

In tables, $\star$ stands for TransBoostler results significantly better than TreeBoostler. $\diamond$ indicates that the difference between TransBoostler against RDN-B results is significantly better.

4.4.1 Depth-first mapping

Tables 4.4, 4.5, and 4.6 present the transfer experiments for pairs IMDB and Cora, Yeast and Twitter, and NELL Sports and NELL Finances when performing depth-first mapping. Each of them was treated as source and target domains on each turn. TreeBoostler cannot transfer from NELL Finances to NELL Sports as it cannot

Table 4.2: Difference between TreeBoostler and TransBoostler using four similarity metrics when mapping IMDB to Cora using depth-first mapping approach.

		TransBoostler			
	TreeBoostler	Soft Cosine	Euclidean	WMD	RWMD
movie	venue	haswordauthor	author	sameauthor	sameauthor

find useful mappings, which results in learning from scratch. Table 4.7 presents the results for IMDB → UW-CSE for the same mapping approach. We omitted the opposite transferring from UW-CSE to IMDB because is as easy as learning from scratch.

The results show TransBoostler performs comparably or better than the baselines in all experiments except for one for AUC ROC. In general, we can positively answer questions **Q1** and **Q2** for TransBoostler. As can be seen, using pre-trained word vectors to find mappings by similarity did improve runtime for most experiments. The only exception is when transferring between pairs IMDB and Cora. For IMDB → Cora, our algorithm is more time-consuming. Theories learned using IMDB contain three distinct predicates. Two of them have arity one, and one is of arity two. Cora has no predicates of arity one, then only one predicate of arity two is mapped. As can be seen in Table 4.2, TreeBoostler finds *venue* as the best mapping for *movie*, while TransBoostler using Soft Cosine, Euclidean distance, WMD, and RWMD finds *haswordauthor*, *author*, *sameauthor*, and *sameauthor*, respectively. Then, it takes more time to TransBoostler to revise the structure. For the opposite experiment, Cora → IMDB, TransBoostler is competitive to both baselines and finds the same mappings as TreeBoostler: *haswordvenue* is mapped to *movie*, and *haswordtitle* is mapped to *genre*. The other predicates are mapped to empty. This shows mapping by similarity is cogent. TransBoostler also finds cogent mappings when transferring from IMDB to UW-CSE. It finds *actor* and *director* closer to *student* and *professor*, respectively. When using the Euclidean distance, *movie* is mapped to *publication* while other similarity metrics map *movie* to *sameperson*.

When transferring from Yeast to Twitter, we have the same AUC ROC values for all similarity metrics. The source structure contains five distinct predicates, and there are two predicates in the target domain. In this case, all similarity metrics find *interaction* most similar to *follows*, and *location* most similar to *tweets*. It takes more time to learn from the target dataset when using RWMD because it takes 27 seconds to compute similarities. For the experiment Twitter → Yeast, TransBoostler takes far less runtime than TreeBoostler. The source structure contains only two distinct predicates and, as TreeBoostler creates type constraints during mapping to reduce the searching space, only one is mapped to a target predicate. TransBoostler maps both source predicates, as it focuses only on similarity. Then, mapping more

Table 4.3: Difference between TreeBoostler and TransBoostler using four similarity metrics when mapping Twitter to Yeast using depth-first mapping approach.

		TransBoostler			
	TreeBoostler	Soft Cosine	Euclidean	WMD	RWMD
follows	interaction	location	complex	complex	complex
tweets		enzyme	enzyme	interaction	interaction

predicates to a target reduces revision time. Table 4.3 shows the most similar target predicates for each similarity metric.

Lastly, for NELL Sports → NELL Finances, our algorithm underperforms TreeBoostler for AUC ROC. It has a better performance than both baselines when using Soft Cosine and WMD for AUC PR. In this case, TreeBoostler finds four adequate mappings and TransBoostler maps every source predicate to different targets, except for one predicate when using Euclidean distance. It also outperforms RDN-B for both measures and it is more time-consuming. Then, we can still affirmably answer question **Q2** as most of the experiments have competitive or better results than the baselines. In the opposite experiment, NELL Finances → NELL Sports, TransBoostler is more time-consuming when using Soft Cosine and Euclidean. However, we can positively answer question **Q4** as, for most experiments, TransBoostler proved to be less time-consuming than the baselines. For all similarity metrics, it has competitive results for AUC ROC and impairs performance for AUC PR. The only exception is when using Soft Cosine similarity that has competitive results. When learning both NELL datasets, there are over five predicates in the two source structures. Each source predicate is mapped to its corresponding target. As tables are very confusing, we omitted the comparison between mappings performed by both algorithms' mappings.

Experiments for increasing amounts of target data when using depth-first mapping are presented in Figures 4.1, 4.2, 4.3, 4.4, 4.5, 4.6, and 4.7. Results show that TransBoostler outperforms or equates the baselines for AUC ROC and AUC PR curves presented in Figures 4.1, 4.3, and 4.4. Our algorithm underperforms both baselines when transferring from IMDB to Cora as presented in Figure 4.2. When transferring from Twitter to Yeast, TransBoostler has a poor performance when compared to TreeBoostler, but it outperforms or equates RDN-B for most of the amounts of target data. For AUC PR, Figure 4.6 shows it outperforms TreeBooster for most of the similarity metrics for NELL Sports → NELL Finances. It also outperforms RDN-B when using WMD and up to 60% of data. TransBoostler also underperforms RDN-B when transferring from NELL Finances to NELL Sports. In resume, TransBoostler underperforms in experiments where it has difficulty finding the best mappings: IMDB and Cora, NELL Sports and NELL Finances.

Table 4.4: Comparison between TransBoostler and baselines for IMDB and Cora datasets when performing depth-first mapping.

	IMDB → Cora				Cora → IMDB			
	CLL	AUC ROC	AUC PR	Run-time(s)	CLL	AUC ROC	AUC PR	Run-time(s)
RDN-B	-0.693	0.558	0.426	76.97	-0.075	1.000	1.000	2.89
TreeBoostler	-0.659	0.606	0.530	45.74	-0.075	0.999	0.954	4.29
TransBoostler Soft Cosine	-0.675⋆	0.599	0.464	51.18	-0.074	1.000⋆	1.000	4.36
TransBoostler Euclidean	-0.677	0.589	0.453	52.61	-0.076	0.999	0.927	4.42
TransBoostler WMD	-0.668	0.600	0.463	54.44	-0.076	0.999⋆	0.948	4.43
TransBoostler RWMD	-0.662	0.603◇	0.464	76.08	-0.074	1.000⋆◇	0.989	30.18
TreeBoostler*	-0.659	0.574	0.518	1.63	-0.115	0.982	0.888	0.95
TransBoostler* Soft Cosine	-0.699⋆	0.500	0.379	2.20	-0.306⋆◇	0.868	0.092	1.94
TransBoostler* Euclidean	-0.699⋆	0.500	0.379	2.15	-0.304⋆◇	0.868	0.092	1.90
TransBoostler* WMD	-0.699⋆	0.500	0.379	2.23	-0.308⋆◇	0.868	0.092	1.92
TransBoostler* RWMD	-0.699	0.500	0.379	22.18	-0.391⋆◇	0.500	0.026	26.93

Table 4.5: Comparison between TransBoostler and baselines for Yeast and Twitter datasets when performing depth-first mapping.

	Yeast → Twitter				Twitter → Yeast			
	CLL	AUC ROC	AUC PR	Run-time(s)	CLL	AUC ROC	AUC PR	Run-time(s)
RDN-B	-0.122	0.990	0.347	23.45	-0.253	0.926	0.230	15.55
TreeBoostler	-0.096	0.994	0.395	86.63	-0.166	0.986	0.267	34.96
TransBoostler Soft Cosine	-0.127	0.994⋆	0.382	23.38	-0.280⋆◇	0.920	0.169	20.39
TransBoostler Euclidean	-0.107	0.994	0.389	26.09	-0.282⋆◇	0.894	0.325	13.05
TransBoostler WMD	-0.107	0.994	0.374	24.55	-0.240⋆	0.953	0.282	19.64
TransBoostler RWMD	-0.101⋆	0.994	0.400	84.73	-0.224⋆	0.965	0.344⋆	34.11
TreeBoostler*	-0.103	0.993	0.334	7.17	-0.166	0.986	0.267	2.17
TransBoostler* Soft Cosine	-0.154	0.993	0.339	4.51	-0.336⋆◇	0.820	0.299	3.35
TransBoostler* Euclidean	-0.110	0.994	0.405	5.65	-0.336⋆◇	0.820	0.307	2.44
TransBoostler* WMD	-0.110	0.994	0.391	4.45	-0.336⋆◇	0.820	0.304	2.51
TransBoostler* RWMD	-0.110	0.994⋆	0.388⋆	30.20	-0.334⋆◇	0.820	0.310⋆	11.87

Table 4.6: Comparison between TransBoostler and baselines for NELL Sports and NELL Finances datasets when performing depth-first mapping.

	NELL Sports → NELL Finances				NELL Finances → NELL Sports			
	CLL	AUC ROC	AUC PR	Run-time(s)	CLL	AUC ROC	AUC PR	Run-time(s)
RDN-B	-0.323	0.692	0.062	24.86	-0.084	0.993	0.325	303.07
TreeBoostler	-0.165	0.980	0.071	124.59	NA	NA	NA	NA
TransBoostler Soft Cosine	-0.321⋆	0.721	0.087	62.70	-0.117	0.993	0.316	331.29
TransBoostler Euclidean	-0.320⋆	0.750	0.069	54.04	-0.137	0.947	0.248	350.76
TransBoostler WMD	-0.324⋆◇	0.741◇	0.079	53.81	-0.136	0.948	0.243	298.12
TransBoostler RWMD	-0.325⋆	0.713	0.071	93.78	-0.085	0.993	0.276	298.07
TreeBoostler*	-0.315	0.979	0.068	8.85	NA	NA	NA	NA
TransBoostler* Soft Cosine	-0.366⋆◇	0.531	0.001	6.92	-0.372◇	0.484	0.002	12.93
TransBoostler* Euclidean	-0.365⋆◇	0.558	0.002	5.88	-0.375◇	0.488	0.002	13.63
TransBoostler* WMD	-0.365⋆◇	0.540	0.002	5.66	-0.370◇	0.486	0.002	14.02
TransBoostler* RWMD	-0.365⋆◇	0.540	0.002	41.66	-0.368◇	0.494	0.002	48.99

Table 4.7: Comparison between TransBoostler and baselines for pair of datasets IMDB → UW-CSE independent of the mapping approach.

	IMDB → UW-CSE			
	CLL	AUC ROC	AUC PR	Run-time(s)
RDN-B	-0.257	0.940	0.282	8.74
TreeBoostler	-0.247	0.939	0.302	4.78
TransBoostler Soft Cosine	-0.255	0.936	0.284	5.94
TransBoostler Euclidean	-0.254	0.936	0.275	6.44
TransBoostler WMD	-0.247	0.936	0.274	5.89
TransBoostler RWMD	-0.250	0.938	0.281	27.23
TreeBoostler*	-0.267	0.930	0.293	0.63
TransBoostler* Soft Cosine	-0.385⋆◇	0.608	0.035	1.17
TransBoostler* Euclidean	-0.296⋆◇	0.906	0.131	1.53
TransBoostler* WMD	-0.288⋆◇	0.906	0.131	1.19
TransBoostler* RWMD	-0.286	0.906	0.131	21.97

4.4.2 Ranked-first mapping

Tables 4.9 and 4.10 present the results for Yeast → Twitter and pairs NELL Sports and NELL Finances when using ranked-first mapping. Due to IMDB having only one predicate of arity two and two of arity one, there is no difference between mapping approaches when transferring from IMDB to UW-CSE and Cora. For both experiments, *movie* is mapped to its most similar. We omitted the results for Cora → IMDB because it is the same as learning from scratch.

For Yeast → Twitter transfer experiment, Table 4.9 shows improvement for AUC ROC and AUC PR values when using Soft Cosine, WMD and RWMD as similarity metrics. We can answer question **Q1** positively for ranked-first mapping. However, when using Soft Cosine, Euclidean distance, and WMD, it takes more time to learn from the target data when compared to depth-first mapping runtimes. When using RWMD is less time-consuming when compared to the previous approach. We cannot answer question **Q4** affirmably because it is more time-consuming than RDN-B. Table 4.8 shows the difference between mappings when using different similarity metrics. As can be seen, only Soft Cosine maps predicates differently, which might explain why it has better results when compared to others. For the opposite experiment, Twitter → Yeast, there is no difference between mapping approaches.

As can be seen in Table 4.10, for NELL Sports → NELL Finances, mapping predicates to their most similar did improve results when using WMD and RWMD for AUC ROC. However, it impairs performance for AUC PR. It also impairs performance when using Euclidean distance for both AUC ROC and AUC PR. In this case, TransBoostler performing ranked-first is less time-consuming than when performing depth-first mapping and it is still less time-consuming than TreeBoostler (**Q4**). Finally, for NELL Finances → NELL Sports, there is improvement for both

Table 4.8: Difference between TreeBoostler and TransBoostler using four similarity metrics when mapping Yeast to Twitter using ranked-first mapping approach.

		TransBoostler			
	TreeBoostler	Soft Cosine	Euclidean	WMD	RWMD
interaction	follows	tweets	tweets	tweets	tweets
location		follows			
enzyme					
phenotype					
complex			follows	follows	follows

AUC ROC and AUC PR for Euclidean distance and WMD. Ranked-first mapping improves performance for AUC PR except when using Soft Cosine. It also takes less time to learn a new model than when using depth-first mapping (except for RWMD). In general, results are competitive to RDN-B and we do have a faster approach (**Q4**). As mentioned before, we omit the comparison between algorithms' mappings as NELL datasets have too many predicates in their source structures.

Figures 4.8, 4.9, and 4.10 present the results when performing ranked-first mapping for Yeast → Twitter, NELL Sports → NELL Finances, and NELL Finances → NELL Sports, respectively. We can observe the same results for Yeast → Twitter. TransBoostler outperforms or equates both baselines for AUC ROC and AUC PR when increasing the amounts of target data. When transferring from NELL Sports to NELL Finances, it underperforms TreeBoostler and equates RDN-B for AUC ROC. For AUC PR, it underperforms learning from scratch but it has a better performance when compared to TreeBoostler. Finally, for the opposite experiment, it underperforms RDN-B for AUC ROC, except when using 60% of target data for Euclidean distance, WMD and RWMD. It also underperforms the baseline for AUC PR.

4.4.3 Final Remarks

As can be seen in tables, RWMD has shown to be a costly similarity metric. The best runtime is when mapping predicates from Twitter to Yeast in which takes around 10 seconds. It is too much time when compared to other similarity metrics, which take less than a second. For pairs of datasets such as NELL Sports and NELL Finances, it takes 39 and 42 seconds, respectively. Then, our algorithm when relying on RWMD is more time-consuming than the baselines.

Both approaches lead to the same answer to question **Q3**: when compared to TreeBoostler*, our algorithm is more dependent on theory revision in general. It has a poor performance for pairs IMDB and Cora, and NELL Sports and NELL Finances when no revision is applied. Only Yeast → Twitter experiment has com-

Table 4.9: Comparison between TransBoostler and baselines for Yeast and Twitter datasets when performing ranked-first mapping.

	Yeast → Twitter			
	CLL	AUC ROC	AUC PR	Run-time(s)
RDN-B	-0.122	0.990	0.347	23.45
TreeBoostler	-0.096	0.994	0.395	86.63
TransBoostler Soft Cosine	-0.088	0.996	0.465	65.22
TransBoostler Euclidean	-0.099	0.994	0.397	63.98
TransBoostler WMD	-0.098	0.995	0.421◇	51.09
TransBoostler RWMD	-0.098	0.995⋆	0.425	67.20
TreeBoostler*	-0.103	0.993	0.334	7.17
TransBoostler* Soft Cosine	-0.090	0.993	0.339	4.65
TransBoostler* Euclidean	-0.159	0.993	0.341	2.20
TransBoostler* WMD	-0.158	0.993	0.343	2.13
TransBoostler* RWMD	-0.159	0.993	0.343	2.15

petitive results for TransBoostler*. In this case, both datasets have very similar source structures, which might facilitate transference. In the end, revision is really important for our proposed mapping component.

In resume, we do have competitive results for most pairs of datasets. Then, we can positively answer question **Q1** for both mapping approaches as TransBoostler performs equally or better than baselines for most experiments. The results presented in this section show TransBoostler can successfully transfer theories by relying on word embeddings similarity (**Q2**). Also, mapping by similarity is faster than previously transfer learning-based approaches but it can be more time-consuming than learning from scratch (**Q4**).

In general, ranked-first mapping performs better than depth-first mapping as it improves results for AUC ROC for all pairs of experiments. Nevertheless, ranked-first mapping is more time-consuming when transferring from Yeast to Twitter but is less time-consuming for pairs of datasets NELL Sports and NELL Finances. As predicates are mapped accordingly to a rank of similarities, i.e., ignoring the order they appear in the source structure, it might be the reason ranked-first mapping is more time-consuming than depth-first mapping. When performing ranked-first mapping, TransBoostler needs to perform more changes in the tree structure because it causes more literals or inner nodes to be discarded, increasing the revision time. For NELL pairs of datasets, all predicates have a corresponding target except for one. This is the only case there is no runtime increase. Since TransBoostler proposes to perform first-order-based transfer learning, it is reasonable to have a better performance when using ranked-first mapping. NELL datasets corroborate this assumption as their predicates have more semantic values. All predicates are small phrases.

Table 4.10: Comparison between TransBoostler and baselines for NELL Sports and NELL Finances datasets when performing ranked-first mapping.

	NELL Sports → NELL Finances				NELL Finances → NELL Sports			
	CLL	AUC ROC	AUC PR	Run-time(s)	CLL	AUC ROC	AUC PR	Run-time(s)
RDN-B	-0.323	0.692	0.062	24.86	-0.084	0.993	0.325	303.80
TreeBoostler	-0.165	0.980	0.071	124.59	NA	NA	NA	NA
TransBoostler Soft Cosine	-0.315⋆	0.728◇	0.081	57.46	-0.130	0.990	0.261	225.84
TransBoostler Euclidean	-0.343⋆◇	0.606	0.051	42.78	-0.138	0.992	0.290	229.65
TransBoostler WMD	-0.213⋆	0.952◇	0.062	51.21	-0.123	0.991	0.304	259.52
TransBoostler RWMD	-0.214⋆	0.952◇	0.054	66.86	-0.144	0.992	0.299	640.87
TreeBoostler*	-0.315	0.979	0.068	8.85	NA	NA	NA	NA
TransBoostler* Soft Cosine	-0.369⋆◇	0.494	0.001	2.22	-0.372◇	0.491	0.002	4.07
TransBoostler* Euclidean	-0.372⋆◇	0.492	0.001	2.32	-0.371◇	0.483	0.002	4.22
TransBoostler* WMD	-0.220◇	0.940◇	0.012	2.50	-0.375◇	0.485	0.002	9.37
TransBoostler* RWMD	-0.368⋆◇	0.541	0.002	46.45	-0.370◇	0.489	0.002	14.37

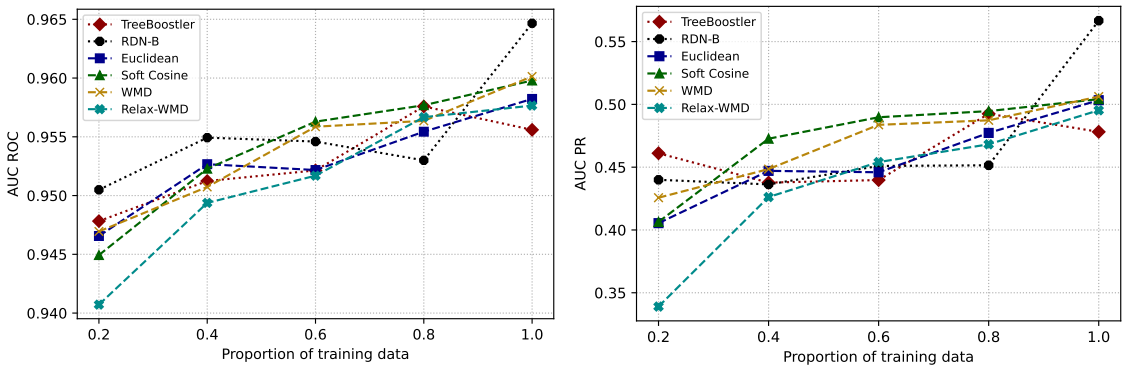

Figure 4.1: Learning curves for AUC ROC (left) and AUC PR (right) for IMDB → UW-CSE transfer experiment when performing depth-first mapping.

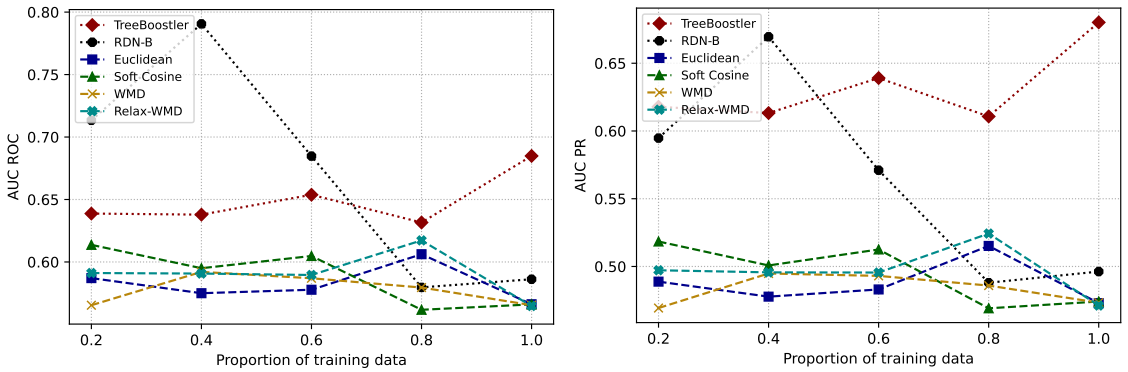

Figure 4.2: Learning curves for AUC ROC (left) and AUC PR (right) for IMDB → Cora transfer experiment when performing depth-first mapping.

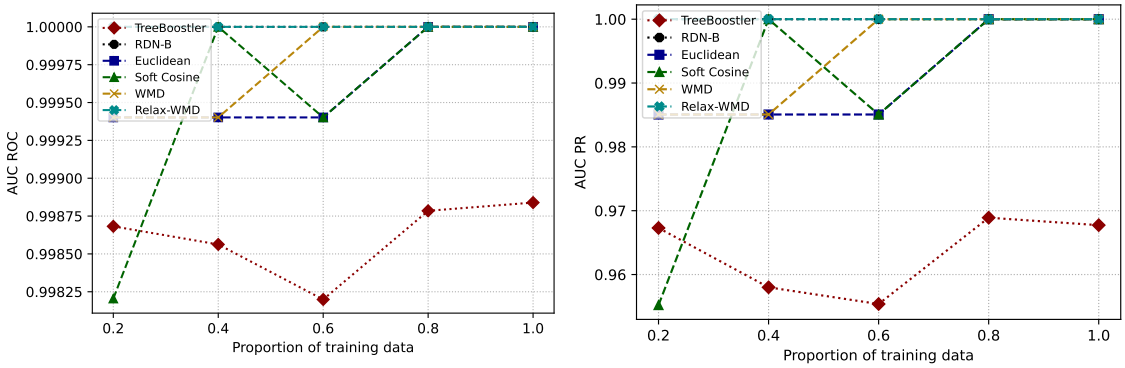

Figure 4.3: Learning curves for AUC ROC (left) and AUC PR (right) for Cora → IMDB transfer experiment when performing depth-first mapping.

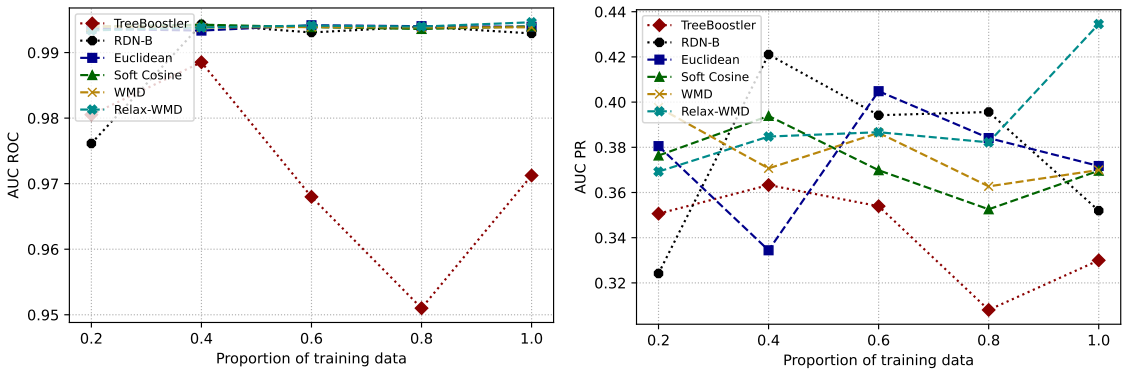

Figure 4.4: Learning curves for AUC ROC (left) and AUC PR (right) for Yeast → Twitter transfer experiment when performing depth-first mapping.

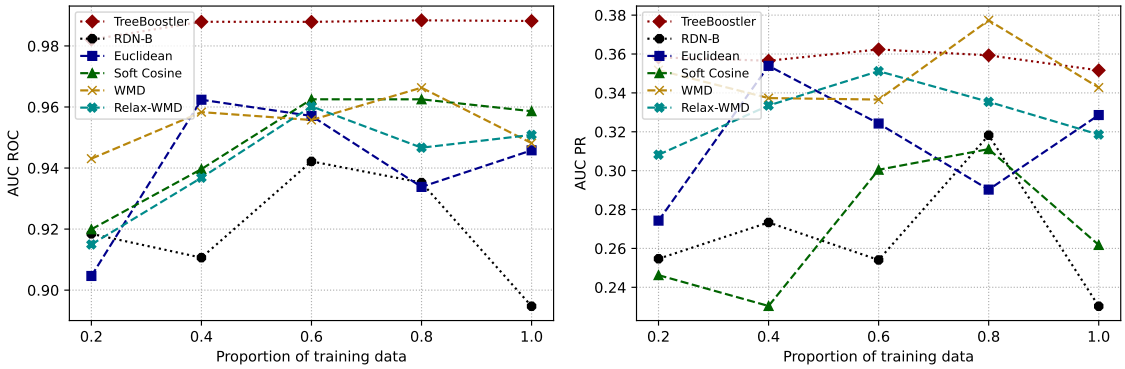

Figure 4.5: Learning curves for AUC ROC (left) and AUC PR (right) for Twitter → Yeast transfer experiment when performing depth-first mapping.

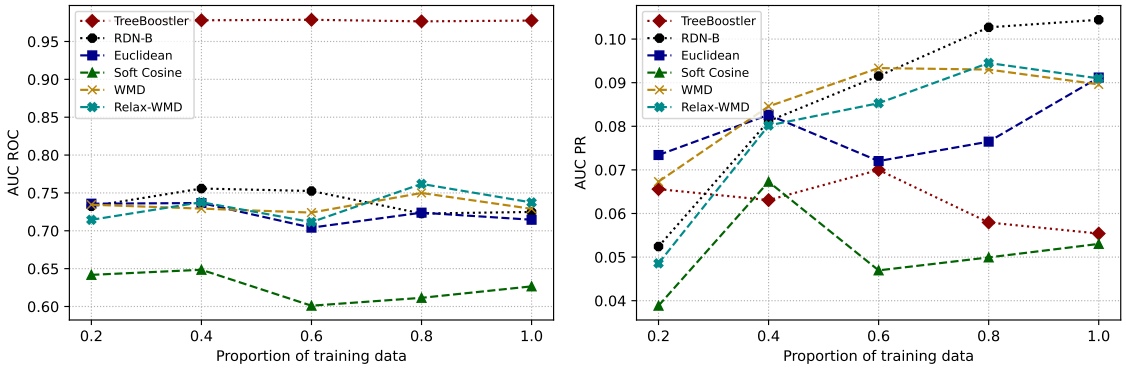

Figure 4.6: Learning curves for AUC ROC (left) and AUC PR (right) for NELL Sports → NELL Finances transfer experiment when performing depth-first mapping.

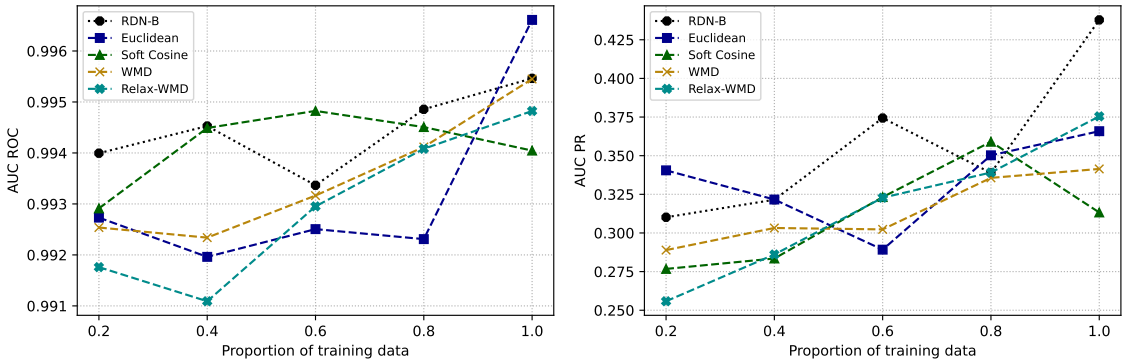

Figure 4.7: Learning curves for AUC ROC (left) and AUC PR (right) for NELL Finances → NELL Sports transfer experiment when performing depth-first mapping.

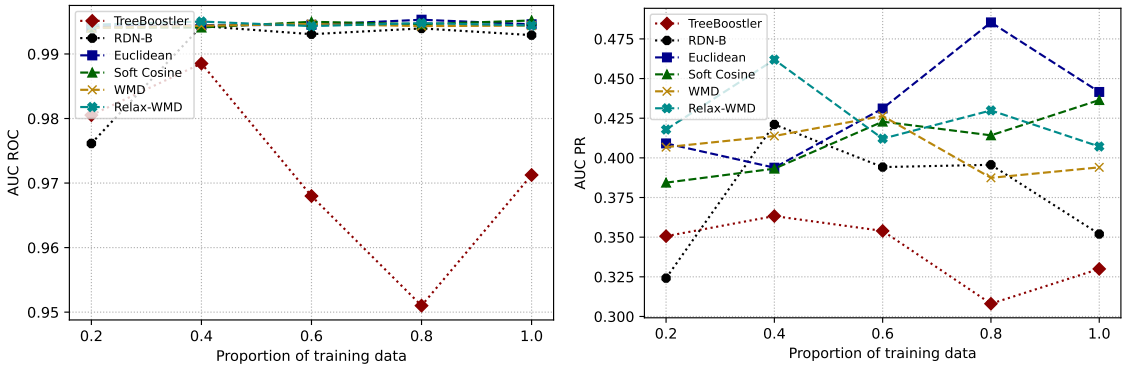

Figure 4.8: Learning curves for AUC ROC (left) and AUC PR (right) for Yeast → Twitter transfer experiment when performing ranked-first mapping.

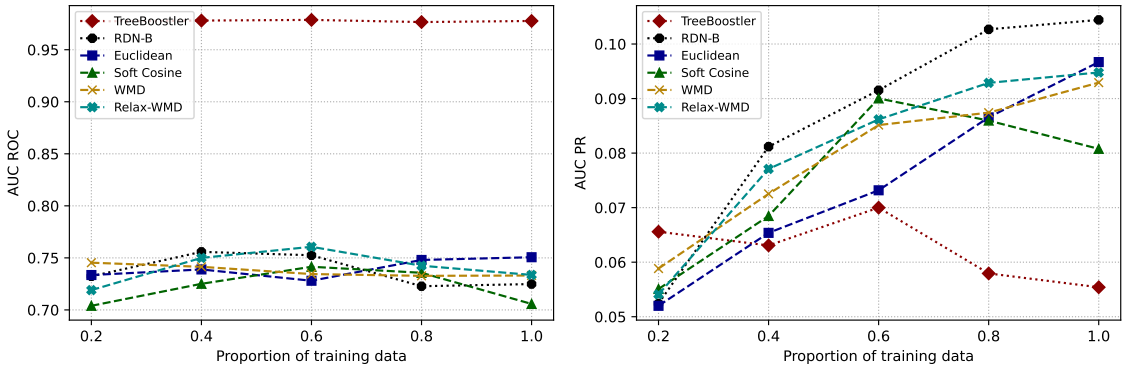

Figure 4.9: Learning curves for AUC ROC (left) and AUC PR (right) for NELL Sports → NELL Finances transfer experiment when performing ranked-first mapping.

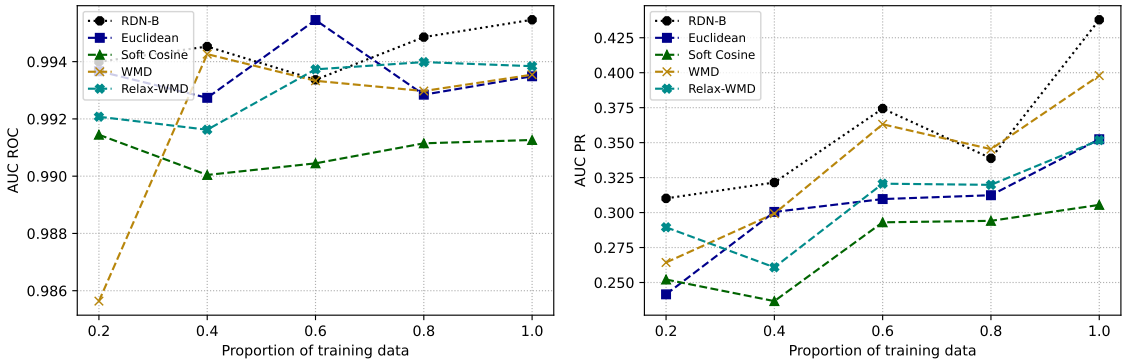

Figure 4.10: Learning curves for AUC ROC (left) and AUC PR (right) for NELL Finances → NELL Sports transfer experiment when performing ranked-first mapping.

Chapter 5

Conclusion

In this dissertation, we presented TransBoostler: an algorithm that transfers Boosted RDNs learned from a source domain to a different target domain. Given the structure of the source regression trees, it constructs a target set of regression trees as it leverages pre-trained word embeddings to find mappings between predicates by similarity. We propose two mapping approaches: the first one follows the order predicates appear in the structure, then, predicates closer to the root have priority and; the second one ignores such order and maps predicates accordingly to a rank of the most similar pairs of *(source, target)* predicates. It also relies on theory revision to the mapped model by pruning and expanding nodes in order to improve its accuracy.

We have performed a set of experiments to evaluate TransBoostler using six publicy available datasets and four similarity metrics. First, we simulate a transfer learning scenario where only a few data are available to investigate if TransBoostler can successfully transfer across different domains. As observed in the experimental results, mapping by similarity has a good performance and can be less time-consuming than a previous related transfer learning approach, but depends on the pair of datasets and the similarity metric used. When compared to learning from scratch, it improves performance but it can be more time-consuming. Then, theory revision has proved to be a very important process, as just transferring the structure and parameter learning have worse performance for most pairs of experiments. The only exception is the pair of datasets Yeast and Twitter. For both transferring experiments, we have competitive results even when theory revision is not applied. In this case, both datasets have very similar source structures, which might facilitate transference. Experiments also showed that theory revision is an effort that considerably raises training time.

Experimental results also showed that the order in which predicates are mapped matters. For some pairs of datasets and similarity metrics, the performance did improve when using ranked-first mapping. For other pairs, the depth-first mapping

leads to the best performance. The best order to follow when mapping depends on the pairs of datasets and the similarity metric chosen. Regardless the mapping approach, for most experiments, it is better to use TransBoostler as it performs equally or better than TreeBoostler in less runtime. It also learns more accurate models than RDN-B, even though it is more time-consuming. However, for the pair IMDB and Cora it is still more guaranteed to search the mapping space.

5.1 Future Works

There are many possible future work directions. First, one could generalize the mapping component proposed in this dissertation to select the top-N most similar predicates to proceed with the mapping. Given a list of the top-N most similar predicates, we can use the weighted variance to choose the predicate that gives the node the best split.

It remains a future investigation to understand whether or not to transfer from one domain to another and the effect of the data in which the embeddings were trained. As observed in [59], results when using word embeddings must depend on the embedding method used. Then, another possible future work direction is to explore word embeddings methods and contexts.

The Hungarian algorithm [72] can also be applied to find the optimal solution to the Maximum-Weight Bipartite Matching problem. As we use a greedy approach, it only approximates the optimal solution. Another possible research question is combining the types of arguments with predicates to try to find better mappings or using more information like the height of a predicate in the tree, for example. A simpler approach is to test different similarity metrics.

Finally, the proposed mapping component in this dissertation can also be applied to different and more general relational models. Then, testing this component to other SRL models is also interesting to investigate.

References

[1] YANG, Q., ZHANG, Y., DAI, W., et al. *Transfer Learning*. Cambridge University Press, 2020. ISBN: 9781107016903. doi: 10.1017/9781139061773.

[2] NATARAJAN, S., KHOT, T., KERSTING, K., et al. "Gradient-based boosting for statistical relational learning: The relational dependency network case", *Machine Learning*, v. 86, n. 1, pp. 25–56, 2012.

[3] AZEVEDO SANTOS, R., PAES, A., ZAVERUCHA, G. "Transfer learning by mapping and revising boosted relational dependency networks", *Machine Learning*, v. 109, pp. 1435–1463, 2020.

[4] MITCHELL, T. M. *Machine learning, International Edition*. McGraw-Hill Series in Computer Science. McGraw-Hill, 1997. ISBN: 978-0-07-042807-2.

[5] HIRSCH, S., GUY, I., NUS, A., et al. "Query reformulation in E-commerce search". In: *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 1319–1328, 2020.

[6] TAN, C., SUN, F., KONG, T., et al. "A survey on deep transfer learning". In: *International conference on artificial neural networks*, pp. 270–279. Springer, 2018.

[7] WU, Z., ZHAO, D., LIANG, Q., et al. "Dynamic sparsity neural networks for automatic speech recognition". In: *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 6014–6018. IEEE, 2021.

[8] DE RAEDT, L. *Logical and relational learning*. Springer Science & Business Media, 2008.

[9] KHOSRAVI, H., BINA, B. "A survey on statistical relational learning". In: *Canadian conference on artificial intelligence*, pp. 256–268. Springer, 2010.

[10] GETOOR, L., TASKAR, B. *Introduction to Statistical Relational Learning (Adaptive Computation and Machine Learning)*. The MIT Press, 2007. ISBN: 0262072882.

[11] TORREY, L., SHAVLIK, J. "Transfer learning". In: *Handbook of research on machine learning applications and trends: algorithms, methods, and techniques*, pp. 242–264. IGI global, 2010.

[12] KUMARASWAMY, R., ODOM, P., KERSTING, K., et al. "Transfer learning via relational type matching". In: *2015 IEEE International Conference on Data Mining*, pp. 811–816. IEEE, 2015.

[13] 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, p. 608–614. AAAI Press, 2007. ISBN: 9781577353232.

[14] 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, p. 1300–1307, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc.

[15] PILEHVAR, M. T., CAMACHO-COLLADOS, J. "Embeddings in Natural Language Processing: Theory and Advances in Vector Representations of Meaning", *Synthesis Lectures on Human Language Technologies*, v. 13, n. 4, pp. 1–175, 2020. doi: 10.2200/S01057ED1V01Y202009HLT047.

[16] WROBEL, S. "First Order Theory Refinement". In: De Raedt, L. (Ed.), *Advances in Inductive Logic Programming*, IOS Press, 1996.

[17] LUCA, T., PAES, A., ZAVERUCHA, G. "Mapping Across Relational Domains for Transfer Learning with Word Embeddings-based Similarity". In: *Proceedings of the 30th International Conference on Inductive Logic Programming @ IJCLR, ILP2020-21. Virtual.* Springer, 2021.

[18] MEWES, H.-W., FRISHMAN, D., GÜLDENER, U., et al. "MIPS: a database for genomes and protein sequences", *Nucleic acids research*, v. 30, n. 1, pp. 31–34, 2002.

[19] LEVESQUE, H. J. "Knowledge representation and reasoning", *Annual review of computer science*, v. 1, n. 1, pp. 255–287, 1986.

[20] RUSSELL, S., NORVIG, P. *Artificial Intelligence: A Modern Approach*. 3rd ed. USA, Prentice Hall Press, 2009. ISBN: 0136042597.

[21] BRATKO, I. *PROLOG Programming for Artificial Intelligence*. 2nd ed. USA, Addison-Wesley Longman Publishing Co., Inc., 1990. ISBN: 0201416069.

[22] NIENHUYS-CHENG, S.-H., DE WOLF, R. *Foundations of inductive logic programming*, v. 1228. Springer Science & Business Media, 1997.

[23] MUGGLETON, S., DE RAEDT, L. "Inductive Logic Programming: Theory and methods", *The Journal of Logic Programming*, v. 19-20, pp. 629–679, 1994. ISSN: 0743-1066. doi: https://doi.org/10.1016/0743-1066(94)90035-3. Special Issue: Ten Years of Logic Programming.

[24] GETOOR, L., FRIEDMAN, N., KOLLER, D., et al. "Probabilistic relational models", *Introduction to statistical relational learning*, v. 8, 2007.

[25] NEVILLE, J., JENSEN, D. "Relational dependency networks." *Journal of Machine Learning Research*, v. 8, n. 3, 2007.

[26] KERSTING, K., DE RAEDT, L. "1 Bayesian logic programming: theory and tool", *Statistical Relational Learning*, p. 291, 2007.

[27] RICHARDSON, M., DOMINGOS, P. "Markov logic networks", *Machine learning*, v. 62, n. 1-2, pp. 107–136, 2006.

[28] HECKERMAN, D. "A tutorial on learning with Bayesian networks", *Innovations in Bayesian networks*, pp. 33–82, 2008.

[29] POOLE, D. "Probabilistic Horn abduction and Bayesian networks", *Artificial intelligence*, v. 64, n. 1, pp. 81–129, 1993.

[30] DE RAEDT, L., DEHASPE, L. "Clausal discovery", *Machine Learning*, v. 26, n. 2, pp. 99–146, 1997.

[31] HECKERMAN, D., CHICKERING, D. M., MEEK, C., et al. "Dependency networks for inference, collaborative filtering, and data visualization", *Journal of Machine Learning Research*, v. 1, n. Oct, pp. 49–75, 2000.

[32] 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*, pp. 625–630, 2003.

[33] NEVILLE, J., JENSEN, D., GALLAGHER, B. "Simple estimators for relational Bayesian classifiers". In: *Third IEEE International Conference on Data Mining*, pp. 609–612, 2003. doi: 10.1109/ICDM.2003.1250989.

[34] FRIEDMAN, J. H. "Greedy function approximation: a gradient boosting machine", *Annals of statistics*, pp. 1189–1232, 2001.

[35] 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*, p. 28, 2004.

[36] GUTMANN, B., KERSTING, K. "TildeCRF: Conditional random fields for logical sequences". In: *European Conference on Machine Learning*, pp. 174–185. Springer, 2006.

[37] BLOCKEEL, H., DE RAEDT, L. "Top-down induction of first-order logical decision trees", *Artificial intelligence*, v. 101, n. 1-2, pp. 285–297, 1998.

[38] PAN, S. J., YANG, Q. "A survey on transfer learning", *IEEE Transactions on knowledge and data engineering*, v. 22, n. 10, pp. 1345–1359, 2009.

[39] RAINA, R., BATTLE, A., LEE, H., et al. "Self-taught learning: transfer learning from unlabeled data". In: *Proceedings of the 24th international conference on Machine learning*, pp. 759–766, 2007.

[40] SALTON, G., WONG, A., YANG, C.-S. "A vector space model for automatic indexing", *Communications of the ACM*, v. 18, n. 11, pp. 613–620, 1975.

[41] TORREGROSSA, F., ALLESIARDO, R., CLAVEAU, V., et al. "A survey on training and evaluation of word embeddings", *International Journal of Data Science and Analytics*, pp. 1–19, 2021.

[42] MIKOLOV, T., CHEN, K., CORRADO, G., et al. "Efficient Estimation of Word Representations in Vector Space". In: Bengio, Y., LeCun, Y. (Eds.), *1st International Conference on Learning Representations, ICLR 2013, Scottsdale, Arizona, USA, May 2-4, 2013, Workshop Track Proceedings*, 2013.

[43] BOJANOWSKI, P., GRAVE, E., JOULIN, A., et al. "Enriching word vectors with subword information", *Transactions of the Association for Computational Linguistics*, v. 5, pp. 135–146, 2017.

[44] MIKOLOV, T., SUTSKEVER, I., CHEN, K., et al. "Distributed Representations of Words and Phrases and their Compositionality". In: Burges, C. J. C., Bottou, L., Welling, M., et al. (Eds.), *Advances in Neural Information Processing Systems*, v. 26. Curran Associates, Inc., 2013.

[45] SIDOROV, G., GELBUKH, A., GOMEZ ADORNO, H., et al. "Soft Similarity and Soft Cosine Measure: Similarity of Features in Vector Space Model", *Computación y Sistemas*, v. 18, 09 2014. doi: 10.13053/cys-18-3-2043.

[46] KUSNER, M., SUN, Y., KOLKIN, N., et al. "From Word Embeddings To Document Distances". In: Bach, F., Blei, D. (Eds.), *Proceedings of the 32nd International Conference on Machine Learning*, v. 37, *Proceedings of Machine Learning Research*, pp. 957–966, Lille, France, 07–09 Jul 2015. PMLR.

[47] RUBNER, Y., TOMASI, C., GUIBAS, L. J. "A metric for distributions with applications to image databases". In: *Sixth International Conference on Computer Vision (IEEE Cat. No. 98CH36271)*, pp. 59–66. IEEE, 1998.

[48] PELE, O., WERMAN, M. "Fast and robust earth mover's distances". In: *2009 IEEE 12th international conference on computer vision*, pp. 460–467. IEEE, 2009.

[49] TANG, X., LI, Y., SUN, Y., et al. "Transferring Robustness for Graph Neural Network Against Poisoning Attacks". In: *Proceedings of the 13th International Conference on Web Search and Data Mining*, p. 600–608, New York, NY, USA, Association for Computing Machinery, 2020. ISBN: 9781450368223.

[50] HAN, X., HUANG, Z., AN, B., et al. "Adaptive Transfer Learning on Graph Neural Networks". In: *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, p. 565–574, New York, NY, USA, Association for Computing Machinery, 2021. ISBN: 9781450383325.

[51] LEE, C.-K., LU, C., YU, Y., et al. "Transfer learning with graph neural networks for optoelectronic properties of conjugated oligomers", *The Journal of Chemical Physics*, v. 154, n. 2, pp. 024906, 2021.

[52] 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.

[53] MIHALKOVA, L., MOONEY, R. J. "Transfer learning from minimal target data by mapping across relational domains". In: *Twenty-First International Joint Conference on Artificial Intelligence*. Citeseer, 2009.

[54] FIGUEIREDO, L., PAES, A., ZAVERUCHA, G. "Learning for Boosted Relational Dependency Networks Through a Genetic Algorithm". In: *Proceed-*

ings of the 30th International Conference on Inductive Logic Programming @ IJCLR, ILP2020-21. Virtual. Springer, 2021.

[55] VAN HAAREN, J., KOLOBOV, A., DAVIS, J. "TODTLER: Two-Order-Deep Transfer Learning", *Proceedings of the AAAI Conference on Artificial Intelligence*, v. 29, 2015.

[56] BORDES, A., USUNIER, N., GARCIA-DURAN, A., et al. "Translating embeddings for modeling multi-relational data". In: *Neural Information Processing Systems (NIPS)*, pp. 1–9, 2013.

[57] WANG, Z., ZHANG, J., FENG, J., et al. "Knowledge Graph Embedding by Translating on Hyperplanes". In: *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence*, AAAI'14, p. 1112–1119. AAAI Press, 2014.

[58] LIN, Y., LIU, Z., SUN, M., et al. "Learning Entity and Relation Embeddings for Knowledge Graph Completion". In: *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, AAAI'15, p. 2181–2187. AAAI Press, 2015. ISBN: 0262511290.

[59] VIG, L., SRINIVASAN, A., BAIN, M., et al. "An investigation into the role of domain-knowledge on the use of embeddings". In: *International Conference on Inductive Logic Programming*, pp. 169–183. Springer, 2017.

[60] BAZIOTIS, C., PELEKIS, N., DOULKERIDIS, C. "DataStories at SemEval-2017 Task 4: Deep LSTM with Attention for Message-level and Topic-based Sentiment Analysis". In: *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*, pp. 747–754, Vancouver, Canada, August 2017. Association for Computational Linguistics.

[61] FORNEY, G. "The viterbi algorithm", *Proceedings of the IEEE*, v. 61, n. 3, pp. 268–278, 1973. doi: 10.1109/PROC.1973.9030.

[62] JURAFSKY, D., MARTIN, J. H. *Speech and language processing : an introduction to natural language processing, computational linguistics, and speech recognition*. Upper Saddle River, N.J., Pearson Prentice Hall, 2009. ISBN: 9780131873216 0131873210.

[63] MILLER, G. A. "WordNet: a lexical database for English", *Communications of the ACM*, v. 38, n. 11, pp. 39–41, 1995.

[64] TOUTANOVA, K., KLEIN, D., MANNING, C. D., et al. "Feature-Rich Part-of-Speech Tagging with a Cyclic Dependency Network". In: *Proceedings*

of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology - Volume 1, NAACL '03, p. 173–180, USA, 2003. Association for Computational Linguistics. doi: 10.3115/1073445.1073478.

[65] MIKOLOV, T., GRAVE, E., BOJANOWSKI, P., et al. "Advances in Pre-Training Distributed Word Representations". In: *Proceedings of the International Conference on Language Resources and Evaluation (LREC 2018)*, 2018.

[66] MIHALKOVA, L., MOONEY, R. J. "Bottom-up Learning of Markov Logic Network Structure". In: *Proceedings of the 24th International Conference on Machine Learning*, ICML '07, p. 625–632, New York, NY, USA, 2007. Association for Computing Machinery. ISBN: 9781595937933. doi: 10.1145/1273496.1273575.

[67] MIHALKOVA, L., MOONEY, R. "Transfer Learning with Markov Logic Networks". In: *Proceedings of the ICML-06 Workshop on Structural Knowledge Transfer for Machine Learning*, Pittsburgh, PA, June 2006. Disponível em: <http://www.cs.utexas.edu/users/ai-lab?mihalkova:icml-wkshp06>.

[68] 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, p. 39–48, New York, NY, USA, 2003. Association for Computing Machinery. ISBN: 1581137370. doi: 10.1145/956750.956759.

[69] 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.

[70] 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, p. 1306–1313. AAAI Press, 2010.

[71] DAVIS, J., GOADRICH, M. "The relationship between Precision-Recall and ROC curves". In: *Proceedings of the 23rd International Conference on Machine learning*, pp. 233–240, 2006.

[72] KUHN, H. W. "The Hungarian method for the assignment problem", *Naval research logistics quarterly*, v. 2, n. 1-2, pp. 83–97, 1955.