TODTLER: Two-Order-Deep Transfer Learning

Jan Van Haaren
Department of Computer Science
KU Leuven, Belgium
jan.vanhaaren@cs.kuleuven.be

Andrey Kolobov
Microsoft Research
Redmond, WA, USA
akolobov@microsoft.com

Jesse Davis
Department of Computer Science
KU Leuven, Belgium
jesse.davis@cs.kuleuven.be

Abstract

The traditional way of obtaining models from data, *inductive learning*, has proved itself both in theory and in many practical applications. However, in domains where data is difficult or expensive to obtain, e.g., medicine, *deep transfer learning* is a more promising technique. It circumvents the model acquisition difficulties caused by scarce data in a *target domain* by carrying over structural properties of a model learned in a *source domain* where training data is ample. Nonetheless, the lack of a principled view of transfer learning so far has limited its adoption. In this paper, we address this issue by regarding transfer learning as a process that biases learning in a target domain in favor of patterns useful in a source domain. Specifically, we consider a first-order logic model of the data as an instantiation of a set of second-order templates. Hence, the usefulness of a model is partly determined by the learner's prior distribution over these template sets. The main insight of our work is that transferring knowledge amounts to acquiring a posterior over the second-order template sets by learning in the source domain and using this posterior when learning in the target setting. Our experimental evaluation demonstrates our approach to outperform the existing transfer learning techniques in terms of accuracy and runtime.

Introduction

Most research in building models from data has focused on the paradigm of *inductive learning*. In this paradigm, a learner tries to generalize the available training instances in order to classify test instances from the same distribution as the training set. Unfortunately, learning a good model in this way can be difficult if the amount of available training data is small. This is often the case in important problems, from modeling side effects of a drug to predicting terrorist attacks, where data is expensive or even impossible to obtain at will.

On the other hand, humans cope with a lack of training examples quite well by *transferring* knowledge and intuitions from one setting to another, where the former is called the *source domain* and the latter the *target domain*. For instance, scientists have realized that forecasting terrorist activity has analogies to certain models studied in physics. The concepts and theoretical machinery from the field of physics are distinct from those used in the fields of sociology and conflict resolution. Nonetheless, by carrying over the *structural* patterns of the physics models, researchers have managed to build a model for the frequency of terrorist attacks (Clauset and Wiegel 2010).

In this paper, we introduce a principled framework for *deep transfer learning*, along with an approximate procedure that implements it. In contrast to most other transfer learning approaches, which concentrate on *shallow transfer* (Baxter et al. 1995; Banerjee, Liu, and Youngblood 2006) and hence operate on tasks from the same domain (e.g., using information about the reactivity of one chemical to learn about the reactivity of another), our TODTLER framework can perform transfer between completely different problems. Moreover, unlike existing milestone algorithms for deep transfer, DTM (Davis and Domingos 2009) and TAMAR (Mihalkova, Huynh, and Mooney 2007), it offers a principled view of transfer learning, whose insights, when implemented in the proposed approximation scheme, allow it to outperform the state-of-the-art algorithm DTM in a variety of scenarios.

At a high level, TODTLER views knowledge transfer as the process of learning a declarative bias in the source domain and transferring it to the target domain to improve the learning process. More specifically, we concentrate on learning Markov logic networks (MLNs), a flexible model that combines first-order logic with probabilities. We treat an MLN as an instantiation of a set of second-order templates expressible in a language called SOLT. The likelihood of an MLN model is thus partly determined by the learner's prior distribution over the sets of these second-order templates. The main insight of our work is that transferring knowledge amounts to acquiring a posterior over the sets of second-order templates by learning in the source domain and using this posterior when learning in the target setting. As an example, consider the concept of transitivity, which can be expressed as a second-order template $\mathtt{R}(\mathtt{X},\mathtt{Y}) \wedge \mathtt{R}(\mathtt{Y},\mathtt{Z}) \implies \mathtt{R}(\mathtt{X},\mathtt{Z})$. In this clause, R is a predicate variable; therefore, this template is not specific to any domain, although its instantiations, e.g., $\mathtt{Knows}(\mathtt{X},\mathtt{Y}) \wedge \mathtt{Knows}(\mathtt{Y},\mathtt{Z}) \implies \mathtt{Knows}(\mathtt{X},\mathtt{Z})$, are. In our framework, if learning in the source domain reveals instantiations of the transitivity template to contribute to highly likely models, the learning process in the target domain will prefer models with transitive relations as well.

Thus, the contributions of this paper are as follows:

- We introduce TODTLER, which is a principled framework for deep transfer learning, and describe an approximation procedure that uses existing MLN learners to implement TODTLER's insights. Our implementation is available for download.[1]
- We present extensive empirical results for TODTLER on three domains: Yeast, WebKB, and Twitter. These results show that TODTLER's approximation outperforms the state-of-the-art deep transfer learning method DTM, and the state-of-the-art first-order inductive learner LSM (Kok and Domingos 2010). In addition to learning more accurate models, TODTLER is also much faster than DTM.

Background

TODTLER performs transfer learning in *Markov logic networks*, which combine first-order logic with Markov networks. We now review Markov networks, first- and second-order logic, Markov logic networks, and transfer learning.

Markov Networks

Markov networks (Della Pietra, Della Pietra, and Lafferty 1997) represent a joint distribution over a set of propositional variables $\vec{X}$. Markov networks can be represented as a *log-linear* model $P(\vec{X}=\vec{x}) = \frac{1}{Z}\exp(\sum_j w_j f_j(\vec{x}))$, where f_j is a feature, w_j is a weight, and Z is a normalization constant. A feature may be any real-valued function of the variable assignment.

First- and Second-Order Logic

TODTLER uses a subset of first-order logic that is based on three types of symbols: constants, variables, and predicates. *Constants* (e.g., `Anna`) stand for specific objects in the domain. *Variables* (e.g., `X`), denoted by uppercase letters, range over objects in the domain. *Predicates* (e.g., `Friends(X, Y)`) represent relations among objects. An atom is $\mathtt{P}(\mathtt{s_1}, \ldots, \mathtt{s_n})$, where each $\mathtt{s_i}$ is a constant or variable. A *ground atom* is one in which each $\mathtt{s_i}$ is a constant. A *literal* is an atom or its negation. A *clause* is a disjunction over a finite set of literals.

TODTLER also uses a limited form of second-order logic that allows to augment first-order logic with the notion of predicate variables. For example, in the formula $\forall\, \mathtt{R}, \mathtt{X}, \mathtt{Y}, \mathtt{Z}\, [\, \mathtt{R}(\mathtt{X}, \mathtt{Y}) \wedge \mathtt{R}(\mathtt{Y}, \mathtt{Z}) \implies \mathtt{R}(\mathtt{X}, \mathtt{Z})\,]$, `R` is a predicate variable, to be grounded by a predicate symbol. Thus, second-order logic can express domain-independent relationships, and TODTLER harnesses this ability.

Markov Logic Networks

A Markov logic network (Richardson and Domingos 2006) is a set of pairs (F, w), where F is a formula in first-order logic and w is a real number. Each Markov logic network (MLN) encodes a family of Markov networks. Given a finite set of object constants, an MLN induces a Markov network with one node for each ground atom and one feature for each ground formula of that MLN. The weight of each feature is the weight of the MLN formula that generated that feature. An MLN induces the following probability distribution $\vec{x}$:

$$p(\vec{X}=\vec{x}) = \frac{1}{Z}\exp\left(\sum_{F\in\mathscr{F}} w_F n_F(\vec{x})\right), \qquad (1)$$

where $\mathscr{F}$ is the set of formulas in the MLN, w_F is the weight of formula F, $n_F(\vec{x})$ is the number of true groundings[2] of F in $\vec{x}$, and Z is a normalization constant.

Algorithms have been proposed for learning the weights associated with each MLN formula (e.g., Lowd and Domingos (2007)) as well as the formulas themselves (e.g., Kok and Domingos (2010)). MLN structure learners typically optimize the weighted pseudo-log-likelihood (WPLL) of the data, as opposed to the standard likelihood, since the former is much more efficient to compute.

Transfer Learning

This work falls in the area of *transfer learning*; see Pan and Yang (2010) for an overview. In transfer learning, a machine learning algorithm considers data from another domain, called the source domain, in addition to data from the target domain. More formally, we can define transfer learning as follows:

Given: A target task T, a target domain D_t and a source domain D_s, where $D_s \neq D_t$.

Learn: A target predictive function f_t in D_t using knowledge acquired in D_s.

The key difference between inductive learning and transfer learning is that the latter uses additional data from a source domain to learn a more accurate predictive function.

Unlike most approaches, TODTLER belongs to the class of deep transfer learning methods which are capable of generalizing knowledge between distinct domains. Conceptually, the closest transfer learning approaches to TODTLER are DTM (Davis and Domingos 2009) and TAMAR (Mihalkova, Huynh, and Mooney 2007), which both perform deep transfer in the context of Markov logic networks.

DTM transfers knowledge from one domain to another using second-order cliques, which are sets of literals with predicate variables representing a set of formulas. For example, the clique $\{\ \mathtt{R}(\mathtt{X},\mathtt{Y}),\ \mathtt{R}(\mathtt{Y},\mathtt{X})\ \}$, where `R` is a predicate variable and `X` and `Y` are object variables, gives rise to the second-order formulas

$$\begin{gathered}\mathtt{R}(\mathtt{X},\mathtt{Y}) \wedge \mathtt{R}(\mathtt{Y},\mathtt{X}),\\ \mathtt{R}(\mathtt{X},\mathtt{Y}) \wedge \neg\mathtt{R}(\mathtt{Y},\mathtt{X}),\\ \neg\mathtt{R}(\mathtt{X},\mathtt{Y}) \wedge \mathtt{R}(\mathtt{Y},\mathtt{X}),\\ \neg\mathtt{R}(\mathtt{X},\mathtt{Y}) \wedge \neg\mathtt{R}(\mathtt{Y},\mathtt{X}).\end{gathered}$$

In turn, each of these second-order formulas gives rise to one or multiple first-order formulas.

[1] http://dtai.cs.kuleuven.be/ml/systems/todtler

[2] The number of true groundings of a formula is obtained by counting the number of assignments of constants to variables that make that formula true.

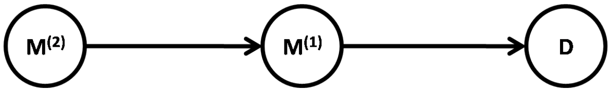


Figure 1: Data generation process

DTM uses the source data to evaluate a number of second-order cliques and transfers a user-defined number of them to the target domain. In the target domain, DTM first considers models only involving the formulas from the transferred cliques and then refines the models to tailor them more to the target domain.

The TAMAR algorithm consists in simply taking a first-order model for the source domain and then attempting to map each clause in the model to the target domain. The algorithm replaces the predicate symbols in each of the clauses with predicates from the target domain in all possible ways. TAMAR is less scalable than DTM or TODTLER in certain scenarios. In particular, exhaustively mapping each source clause to the target domain is time consuming for long clauses or clauses with constants if the target domain has many predicates or many constants.

The field of analogical reasoning (Falkenhainer, Forbus, and Gentner 1989) is also closely related to transfer learning. It applies knowledge from one domain to another via a mapping or correspondence between the objects and relations in the two domains. Often, a human needs to provide possible mappings for each pair of domains.

Transferring Second-Order Knowledge

We now illustrate the intuition behind TODTLER. Suppose we have two datasets, one characterizing the smoking habits of a group of people (the smokers dataset) and the other describing connections between terrorists (the terrorism dataset). An MLN learned on the smokers dataset may contain the first-order clause $\mathtt{Smokes(X)} \wedge \mathtt{Friends(X,Y)} \implies \mathtt{Smokes(Y)}$, which captures the regularity that a friend of a smoker may likely be a smoker. A similar regularity may appear in the terrorism dataset: a person in the same organization with a terrorist may likely be a terrorist. We would like to generalize such regularities from one model to another, but simply transferring first-order clauses does not help because the datasets are described by *different* relationships and properties.

What the domains have in common is the concept of multirelational transitivity described by the *second*-order clause

$$\mathtt{R_1(X)} \wedge \mathtt{R_2(X,Y)} \implies \mathtt{R_1(Y)}, \tag{2}$$

where $\mathtt{R_1}$ and $\mathtt{R_2}$ are predicate variables. It are these types of important structural patterns that TODTLER attempts to identify in the source domain and transfer to other domains. The transfer occurs by *biasing* the learner in the target domain to favor models containing previously discovered regularities in the source domain.

Generative model for the data. More specifically, TODTLER views data in any domain as being generated by the hierarchical process shown in Figure 1. The process starts by producing a second-order model of the data, denoted as $M^{(2)}$. Formally, $M^{(2)}$ is a set of second-order *templates*. Each such template is a clause from a special language called SOLT (**S**econd-**O**rder **L**anguage for **T**emplates), which is a restriction of second-order logic that allows only predicate variables and restricts clause length. Equation 2 provides an example of a SOLT template. SOLT's power stems from being able to use predicate variables in order to state rules that reason about relations, not just objects, and thus describe domain-independent knowledge. $M^{(2)}$ is sampled from a prior $P(M^{(2)})$ induced by independently including each template T expressible in SOLT into $M^{(2)}$ with some probability p_T. In particular, letting $p_T = 0.5$ for every $T \in$ SOLT results in a uniform prior over all possible second-order models.

Given a second-order model $M^{(2)}$, a first-order (MLN) model $M^{(1)}$ is generated by instantiating all templates in $M^{(2)}$ with the set of predicates relevant to the data at hand in all possible ways. Instantiating a template with predicates means grounding each predicate variable in the template with a first-order predicate. In the example above, it is at this stage that the template in Equation 2 gives rise to the first-order formula $\mathtt{Smokes(X)} \wedge \mathtt{Friends(X,Y)} \implies \mathtt{Smokes(Y)}$ and to many others. The weights for the first-order clauses produced in this way, which are necessary for a well-defined MLN, are sampled from some prior probability density (omitted in Figure 1). To complete the process, data is generated from the MLN using a relevant set of object constants to ground the first-order clauses. In the case of the smokers dataset, the data could include ground facts $\mathtt{Friends(Alice,Bob)}$, $\mathtt{Smokes(Alice)}$, and $\mathtt{Smokes(Bob)}$.

Thus, letting a random variable D denote the data, the hierarchical generative model above gives us a joint probability density $p(D, M^{(1)}, M^{(2)})$ that factorizes as

$$p(D, M^{(1)}, M^{(2)}) = \\ p(D|M^{(1)})p(M^{(1)}|M^{(2)})P(M^{(2)}). \tag{3}$$

In this formula, $p(D|M^{(1)})$ is given by Equation 1, $P(M^{(2)})$ is given by the probabilities p_T of including each template T into the second-order model, and $p(M^{(1)}|M^{(2)})$ is positive if the set of clauses in $M^{(1)}$ is the *complete instantiation* of $M^{(2)}$ and 0 otherwise. The set of clauses of an MLN $M^{(1)}$ is the complete instantiation of a second-order model $M^{(2)}$ if this set contains all first-order instantiations of all templates in $M^{(2)}$ (with some formulas possibly having zero weights), and no other formulas.

For the cases when $p(M^{(1)}|M^{(2)}) > 0$, we can write the joint density $p(D, M^{(1)}, M^{(2)})$ in the following form, derived from Equation 1:

Input: D_s — source dataset, D_t — target dataset, $P(M^{(2)})$ — prior over second-order models

Output: $M_t^{(1)^*}$ — an MLN for the target domain

1. Find the posterior distribution $P_s(M^{(2)}|D_s)$ over second-order models such that $P_s(M^{(2)}|D_s)$ is encoded by the set of template probabilities $p_{T,s}$, given the data in the source domain and a similarly encoded prior $P(M^{(2)})$ over second-order models:

$$P_s(M^{(2)}|D_s) \leftarrow \frac{\int_{M_s^{(1)}} p(D_s, M_s^{(1)}|M^{(2)})P(M^{(2)})}{P(D_s)} \tag{5}$$

2. Determine the first-order (MLN) model that maximizes the joint probability of data and first-order model in the target domain if $P_s(M^{(2)}|D_s)$ is used as a prior over second-order models:

$$P_t(M^{(2)}) \leftarrow P_s(M^{(2)}|D_s)$$

$$M_t^{(1)^*} \leftarrow \arg\max_{M^{(1)}} p(D_t|M_t^{(1)}) \sum_{M^{(2)}} p(M_t^{(1)}|M^{(2)})P_t(M^{(2)}) \tag{6}$$

Algorithm 1: The TODTLER framework

$$p(D, M^{(1)}, M^{(2)}) = \prod_{\vec{x}\in D}\left[\frac{1}{Z'}\prod_{T\in\mathscr{T}} p_T \exp\left(\sum_{F\in\mathscr{F}_T} w_F n_F(\vec{x})\right)\right], \tag{4}$$

where p_T is the probability of including template T into second-order model $M^{(2)}$, $\mathscr{F}_T$ is the set of all of T's first-order instantiations, and $\mathscr{T}$ is the set of all second-order templates expressible in SOLT. Equation 4 differs from Equation 1 in only one crucial aspect: the set of all first-order formulas is now divided into disjoint subsets corresponding to particular templates, and each subset has an associated probability p_T of being part of $M^{(1)}$. These probabilities p_T are the key means by which TODTLER transfers knowledge from one domain to another, as we explain next.

Transfer learning with TODTLER. The TODTLER procedure is briefly summarized in Algorithm 1. Let D_s, $M_s^{(1)}$, D_t, and $M_t^{(1)}$stand for the data and first-order MLN in the source domain and in the target domain, respectively. TODTLER performs transfer learning in two steps:

1) Learning the second-order model posterior. In the first step, TODTLER finds the distribution $P_s(M^{(2)}|D_s)$ over second-order models that results from observing the data in the source domain given some initial belief $P(M^{(2)})$ over second-order models (Equation 5 in Algorithm 1). In view of Equation 4, determining $P_s(M^{(2)}|D_s)$ amounts to computing the template probabilities (denoted $p_{T,s}$) according to the source domain data and the prior.

2) Target domain learning using the posterior from the source domain. In the second step, TODTLER determines an MLN $M_t^{(1)^*}$ that maximizes the joint probability of data and first-order model in the target domain *if the posterior $P_s(M^{(2)}|D_s)$ learned in the first step from the source data is used as the prior over second-order models* (Equation 6 in Algorithm 1). In doing so, TODTLER biases model selection for the target dataset by explicitly transferring the learner's experience from the source domain.

Approximations

Despite the conciseness of TODTLER's procedural description (Algorithm 1), implementing it is nontrivial for several reasons. In this section, we discuss the challenges involved and present a series of appropriate approximations to the basic TODTLER framework. Algorithm 2 presents a pseudocode of the resulting implementation, which we will be referring to throughout this section.

Learning second-order model posteriors. The main difficulty presented by TODTLER is computing the posterior distribution $P_s(M^{(2)}|D_s)$ over second-order models. Since we do not assume the distributions in Equation 5 to have any specific convenient form, it is not immediately obvious how to efficiently update the prior $P(M^{(2)})$. Moreover, Equation 5 involves summing over first-order MLNs, suggesting that an exact update procedure would likely be very expensive computationally.

Instead, we take a more heuristic approach. Our procedure exhaustively enumerates all SOLT templates that can form a user-specified maximum clause length L and maximum number of distinct object variables V (line 4). These conditions ensure that the number of second-order templates under consideration is finite and amount to adopting a prior $P(M^{(2)})$ that assigns probability 0 to any second-order model with templates that violate these restrictions. Additionally, we assume that for each template T, its probability of inclusion $p_{T,s}$ under $P_s(M^{(2)}|D_s)$ is correlated with the "usefulness" of the first-order instantiations of T for modeling the data in the source domain and with its prior probability p_T^0.

For each first-order instantiation $F_{T,s}$ in the finite set $\mathscr{F}_S$ of all such instantiations of T generated by replacing T's predicate variables with predicates from the source domain, we calculate F's *usefulness score*, aggregate these numbers across $\mathscr{F}_S$, and use the result, along with the prior p_T^0, as a proxy $\hat{p}_{T,s}$ of $p_{T,s}$. The notion of usefulness of a single first-order formula is fairly crude — each formula typically contributes to the model along with many others, and its effect on the model's performance cannot be easily teased apart

```
1  Input: $D_s$ — source dataset, $D_t$ — target dataset, $L$ — maximum template length, $V$ — maximum number of distinct object variables, $P(M^{(2)}) = \{p_T^0\}_{T \in \text{SOLT}}$ — prior over second-order models
2  Output: $M_t^{(1)^*}$ — an MLN for the target domain
3
4  $\mathscr{T} \leftarrow$ EnumerateSecondOrderTemplates($L$,$V$)
5  $\mathscr{F}_S \leftarrow$ EnumerateFirstOrderFormulas($\mathscr{T}$,$D_s$)
6  $\mathscr{F}_T \leftarrow$ EnumerateFirstOrderFormulas($\mathscr{T}$,$D_t$)
7
8  foreach second-order template $T \in \mathscr{T}$ do
9      foreach first-order formula $F_{T,s} \in \mathscr{F}_S$ do
10         $l_{F,s} \leftarrow$ WPLL of optimal $F_{T,s}$-MLN w.r.t $D_s$
11         $s_{F,s} \leftarrow l_{F,s}$ rescaled to $[0, 1]$
12     end
13     $\hat{p}_{T,s} \leftarrow p_T^0 \cdot average_{F_{T,s} \in \mathscr{F}_S}\{s_{F,s}\}$
14     foreach first-order formula $F_{T,t} \in \mathscr{F}_T$ do
15         $l_{F,t} \leftarrow$ WPLL of optimal $F_{T,t}$-MLN w.r.t $D_t$
16         $s_{F,t} \leftarrow l_{F,t}$ rescaled to $[0, 1]$
17         $\hat{p}_{F,t} \leftarrow s_{F,t} \cdot \hat{p}_{T,s}$
18     end
19 end
20
21 $M_t^{(1)^*} \leftarrow \{\}$
22 $old\_WPLL \leftarrow -\infty$
23 $\mathscr{O}_F \leftarrow$ list of all $F_{T,t} \in \bigcup_{T \in \mathscr{T}} \mathscr{F}_T$ in decreasing order of $\hat{p}_{F,t}$
24
25 foreach first-order formula $F_{T,t} \in \mathscr{O}_F$ do
26     $M_t^{(1)} \leftarrow M_t^{(1)^*} \cup \{F_{T,t}\}$
27     $M_t^{(1)} \leftarrow$ relearn weights
28     if WPLL of $M_t^{(1)}$ w.r.t $D_t > old\_WPLL$ then
29         $old\_WPLL \leftarrow$ WPLL of $M_t^{(1)^*}$ w.r.t $D_t$
30         $M_t^{(1)^*} \leftarrow M_t^{(1)}$
31     end
32 end
33 return $M_t^{(1)^*}$
```

Algorithm 2: An approximation to TODTLER

from that of the rest of the model. Nonetheless, as we explain next, this notion's simplicity also has a big advantage: a formula's usefulness can be computed very efficiently.

More specifically, for each $F_{T,s} \in \mathscr{F}_S$, we perform weight learning in the MLN that contains only the formula $F_{T,s}$. This MLN is denoted as $F_{T,s}$-MLN. The weight learning process makes its own approximations as well. It optimizes the weights so as to maximize the weighted pseudo-log-likelihood (WPLL) of the model, a substitute criterion for optimizing likelihood. That, and the fact that our MLN contains only one formula, makes weight learning very fast. When the learning process finishes, it yields WPLL $l_{F,s}$ of the acquired MLN with respect to the source data (line 10). We then rescale the WPLL to lie between 0 and 1 as different domains can have different ranges of WPLLs, and denote the obtained value as $s_{F,s}$ (line 11).

Intuitively, $s_{F,s}$ roughly reflects how much including $F_{T,s}$ into an MLN helps the model's discriminative power. To estimate the usefulness score $s_{T,s}$ of a template T, we average the clause usefulness scores $s_{F,s}$ across $\mathscr{F}_S$. Thus, we define the approximate sampling probabilities for each template T as $\hat{p}_{T,s} \sim p_T^0 \cdot s_{T,s}$ (line 13).

In the target domain, we similarly compute a probability $\hat{p}_{F,t}$ for each formula $F_{T,t}$, which estimates that formula's probability of inclusion in the target domain model. In a first step, we compute a usefulness score $s_{F,t}$ for each formula $F_{T,t}$ using the same procedure as in the source domain (line 15). In a second step, crucially, we multiply the resulting usefulness score with $\hat{p}_{T,s}$, the posterior probability of inclusion of the corresponding second-order template learned from the source domain (line 17).

Target domain learning. TODTLER builds the target domain model $M_t^{(1)^*}$ incrementally, starting from an empty one in the following way. It arranges the formulas in $\bigcup_{T \in \mathscr{T}} \mathscr{F}_T$ in order of decreasing approximate probability $\hat{p}_{F,t}$ (line 23). For each formula in the ordered list, the approximation procedure attempts to add that formula to $M_t^{(1)^*}$, jointly learning the weights of all already included formulas and computing the model's WPLL (lines 26-27). A formula is added to the model only if doing so increases $M_t^{(1)^*}$'s WPLL with respect to the target data (lines 28-31).

Experimental Evaluation

Our experiments compare the performance of TODTLER to DTM, which is the state-of-the-art transfer learning approach for relational domains (Davis and Domingos 2009). We also compare to learning from scratch using LSM, which is the state-of-the-art MLN structure learning algorithm (Kok and Domingos 2010). We evaluate the performance of the algorithms using data from three domains and address the following four research questions:

- Does TODTLER learn more accurate models than DTM?
- Does TODTLER learn more accurate models than LSM?
- Is TODTLER faster than DTM?
- Does TODTLER discover interesting SOLT templates?

Datasets

We use three datasets of which the first two have been widely used and are publicly available.[3] The **Yeast** protein dataset comes from the MIPS[4] Comprehensive Yeast Genome Database (Mewes et al. 2000; Davis et al. 2005). The dataset includes information about protein location, function, phenotype, class, and enzymes. The **WebKB** dataset consists of labeled web pages from the computer science departments of four universities (Craven and Slattery 2001). The dataset

[3] Available on http://alchemy.cs.washington.edu.

[4] Munich Information Center for Protein Sequence

includes information about links between web pages, words that appear on the web pages, and the classifications of the pages. The **Twitter**[5] dataset contains tweets about Belgian soccer matches. The dataset includes information about follower relations between accounts, words that are tweeted, and the types of the accounts.

Methodology

Each of the datasets is a graph, which is divided into databases consisting of connected sets of facts (Mihalkova, Huynh, and Mooney 2007). Yeast and WebKB consist of four databases while Twitter consists of two. We trained each learner on a subset of the databases and tested it on the remaining databases. We repeated this cycle for all subsets of the databases.

We transferred with both TODTLER and DTM in all six possible source-target settings. Within each domain, both transfer learning algorithms had the same parameter settings. In each domain, we optimized the WPLL for the predicate of interest. We learned the weights of the formulas in the target model using Alchemy (Kok et al. 2010) and applied a pruning threshold of 0.05 on the weights of the clauses.

For DTM, we generated all clauses containing at most three literals and three object variables, and transferred five and ten second-order cliques to the target domain. Since DTM's refinement step can be computationally expensive, we limited its runtime to 100 hours per database.

For TODTLER, we enumerated all second-order templates containing at most three literals and three object variables. We assumed a uniform prior distribution over the second-order templates in the source domain, which means TODTLER's p_T^0 parameter was set to 0.5 for each template.

To evaluate each system, we jointly predict the truth value of all groundings of the *Function* predicate in Yeast, the *PageClass* predicate in WebKB, and the *AccountType* predicate in Twitter given evidence about all other predicates. We computed the probabilities using MC-SAT. After a burn-in of 1,000 samples, we computed the probabilities with the next 10,000 samples.[6] We measured the area under the precision-recall curve (AUCPR) and the test set conditional log-likelihood (CLL) for the predicate of interest. AUCPR gives an indication of the predictive accuracy of the learned model. Furthermore, AUCPR is insensitive to the large number of true negatives in these domains. CLL measures the quality of the probability estimates.

Discussion of Results

Table 1 presents the average relative difference in AUCPR (i.e., $(AUCPR_{\text{TODTLER}} - AUCPR_{DTM})/AUCPR_{\text{TODTLER}}$) and CLL (i.e., $(CLL_{\text{TODTLER}} - CLL_{DTM})/CLL_{\text{TODTLER}}$) between TODTLER and both transferring five (DTM-5) and ten (DTM-10) cliques with DTM. The table shows the average relative differences for varying amounts of training data. For example, the "3 DB" column presents the results for training on all subsets of three databases and evaluating on the remaining database. The N/A entries arise because the Twitter dataset only contains two databases. Positive average relative differences denote settings where TODTLER outperforms DTM. In terms of AUCPR, TODTLER outperforms both DTM-10 and DTM-5 in all 14 settings. In terms of CLL, TODTLER outperforms DTM-10 in 12 of the 14 settings and DTM-5 in 11 of the 14 settings.

Table 2 presents the average relative difference in AUCPR and CLL between TODTLER and LSM, and DTM-5 and LSM. This represents a comparison between performing transfer as opposed to learning from scratch in each domain with the state-of-the-art structure learner LSM.[7] We see that TODTLER also outperforms LSM in all 14 settings in terms of both AUCPR and CLL. DTM-5 outperforms LSM in 12 of the 14 settings in terms of both AUCPR and CLL. In both cases, we see that transferring knowledge from a source task leads to more accurate learned models than simply learning from scratch in the target domain.

Figure 2 presents learning curves for predicting protein function in the Yeast dataset when transferring from WebKB. TODTLER outperforms DTM-10, DTM-5 and LSM. Note that LSM obtains a much worse CLL than the other systems and hence its curve falls out of the range of the graph. Figure 3 presents learning curves for predicting a web page's class in the WebKB dataset when transferring from Twitter. Again, TODTLER exhibits better performance than DTM-10, DTM-5 and LSM. In both figures, TODTLER's performance improves as the amount of training data is increased.

Several possible explanations exist about why TODTLER learns more accurate models than DTM. First, TODTLER transfers fine-grained knowledge because it performs transfer on a per-template basis instead of a per-clique basis. As discussed in the background, DTM's second-order cliques group together multiple second-order formulas. Then, each of these second-order formulas gives rise to one or multiple first-order formulas. Within a clique, only a small subset of these formulas will be helpful for modeling the target domain. Second, TODTLER transfers both the second-order templates (i.e., structural regularities) as well as information about their usefulness (i.e., the posterior of the formulas, $P_s(M^{(2)}|D_s)$). In contrast, DTM just transfers the second-order cliques and the target data is used to assess whether the regularities are important. Finally, TODTLER transfers a more diversified set of regularities whereas DTM is restricted to a smaller set of user-selected cliques. This increases the chance that TODTLER transfers something of use for modeling the target domain.

In addition to learning more accurate models, TODTLER is also considerably faster than DTM (see Table 3). Across all settings, TODTLER is 8 to 44 times faster in Yeast, 5 to 29 times faster in WebKB, and 132 to 264 times faster in Twitter. A couple of reasons contribute to TODTLER's improved runtime. First, for learning in the target domain, DTM runs an iterative greedy strategy that picks the single

[5] Available on http://dtai.cs.kuleuven.be/ml/systems/todtler.

[6] The DTM paper performs leave-one-grounding-out inference while this paper jointly infers all groundings of the target predicate.

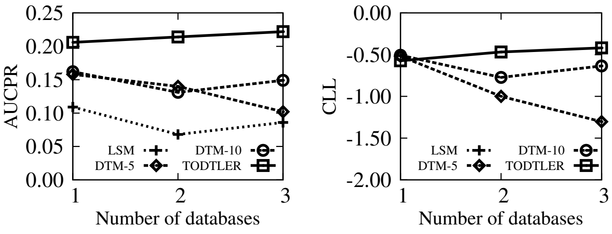
[7] We picked DTM-5 as it generally exhibits better performance than DTM-10.

Table 1: The average relative differences in AUCPR and CLL between TODTLER and DTM-10, and TODTLER and DTM-5 as function of the amount of training data. Positive differences indicate settings where TODTLER outperforms DTM. In terms of AUCPR, TODTLER outperforms both DTM-10 and DTM-5 in all 14 settings. In terms of CLL, TODTLER outperforms DTM-10 in 12 of the 14 settings and DTM-5 in 11 of the 14 settings. The N/A entries arise because the Twitter dataset only contains two databases.

	TODTLER versus DTM-10						**TODTLER versus DTM-5**					
	AUCPR			**CLL**			**AUCPR**			**CLL**		
	1 DB	**2 DB**	**3 DB**	**1 DB**	**2 DB**	**3 DB**	**1 DB**	**2 DB**	**3 DB**	**1 DB**	**2 DB**	**3 DB**
WebKB → Yeast	0.213	0.390	0.331	-0.114	0.650	0.517	0.231	0.344	0.540	-0.097	1.137	2.112
Twitter → Yeast	0.190	0.325	0.362	-0.063	0.126	0.017	0.171	0.353	0.260	-0.065	0.748	-0.127
Yeast → WebKB	0.479	0.614	0.638	0.121	0.098	0.196	0.479	0.607	0.627	0.121	0.095	0.191
Twitter → WebKB	0.578	0.596	0.607	1.055	0.158	0.157	0.578	0.587	0.605	1.055	0.155	0.156
WebKB → Twitter	0.224	N/A	N/A	3.945	N/A	N/A	0.150	N/A	N/A	5.140	N/A	N/A
Yeast → Twitter	0.226	N/A	N/A	4.210	N/A	N/A	0.152	N/A	N/A	5.469	N/A	N/A

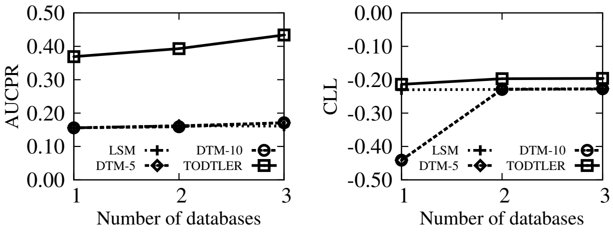
Table 2: The average relative differences in AUCPR and CLL between TODTLER and LSM, and DTM-5 and LSM as function of the amount of training data. Positive differences indicate settings where TODTLER or DTM-5 outperforms LSM. TODTLER outperforms LSM in all 14 settings in terms of both AUCPR and CLL. DTM-5 outperforms LSM in 12 of the 14 settings in terms of both AUCPR and CLL. The N/A entries arise because the Twitter dataset only contains two databases.

	TODTLER versus LSM						**DTM-5 versus LSM**					
	AUCPR			**CLL**			**AUCPR**			**CLL**		
	1 DB	**2 DB**	**3 DB**	**1 DB**	**2 DB**	**3 DB**	**1 DB**	**2 DB**	**3 DB**	**1 DB**	**2 DB**	**3 DB**
WebKB → Yeast	0.471	0.671	0.583	5.075	12.156	8.841	0.311	0.518	0.161	5.725	6.537	3.328
Twitter → Yeast	0.479	0.676	0.589	6.091	14.356	10.486	0.371	0.542	0.459	6.584	8.764	12.832
Yeast → WebKB	0.576	0.561	0.562	0.079	0.073	0.070	0.186	-0.018	0.032	-0.037	0.001	0.006
Twitter → WebKB	0.576	0.561	0.562	0.072	0.066	0.064	-0.004	0.003	0.058	-0.478	0.003	0.007
WebKB → Twitter	0.599	N/A	N/A	13.463	N/A	N/A	0.528	N/A	N/A	1.355	N/A	N/A
Yeast → Twitter	0.600	N/A	N/A	14.238	N/A	N/A	0.528	N/A	N/A	1.355	N/A	N/A



(a) AUCPR in Yeast (b) CLL in Yeast

Figure 2: Learning curves showing AUCPR and CLL for the *Function* predicate in Yeast when transferring from WebKB. The LSM curve falls out of the range of figure (b).



(a) AUCPR in WebKB (b) CLL in WebKB

Figure 3: Learning curves showing AUCPR and CLL for the *PageClass* predicate in WebKB when transferring from Twitter. The LSM and DTM curves largely overlap.

best candidate formula in each step. This is more expensive than TODTLER's non-iterative target-domain strategy for picking formulas. Second, DTM performs a refinement step, which improves accuracy but is computationally costly as it is another greedy search approach.

Table 4 presents the ten top-ranked second-order templates in each domain. One example is $\mathtt{R_1(X,Y)} \vee \neg\mathtt{R_1(Y,X)}$, which represents symmetry and ranks first in Yeast and WebKB and second in Twitter. When transferred to the Twitter problem, this template gives, among others, rise to the first-order formula $\mathtt{Follows(X,Y)} \vee \neg\mathtt{Follows(Y,X)}$, meaning that if an account Y follows an account X, X is likely to follow Y as well. Another example is $\mathtt{R_1(X,Y)} \vee \neg\mathtt{R_1(Z,Y)} \vee \neg\mathtt{R_2(X,Z)}$, which ranks third in Yeast, eighth in WebKB and ninth in Twitter. This template represents the concept of homophily, which means that related objects (X and Z) tend to have similar properties (Y). When transferred to the WebKB problem, this template gives, among others, rise to the first-order formula $\mathtt{Has(X,Y)} \vee \neg\mathtt{Has(Z,Y)} \vee \neg\mathtt{Linked(X,Z)}$, meaning that if a web page X links to a web page Z, both web pages are likely to contain the same word Y.

Table 3: Average runtime in minutes for TODTLER and DTM. TODTLER is consistently faster than both DTM configurations. The N/A entries arise because the Twitter dataset only contains two databases.

	TODTLER			**DTM-10**			**DTM-5**		
	1 DB	**2 DB**	**3 DB**	**1 DB**	**2 DB**	**3 DB**	**1 DB**	**2 DB**	**3 DB**
WebKB → Yeast	103	199	338	1,759	6,671	9,206	1,766	6,234	14,896
Twitter → Yeast	93	181	277	725	1,683	4,635	707	2,496	9,555
Yeast → WebKB	16	26	45	95	571	1,323	84	478	753
Twitter → WebKB	13	21	44	142	392	840	122	294	464
WebKB → Twitter	1	N/A	N/A	75	N/A	N/A	49	N/A	N/A
Yeast → Twitter	1	N/A	N/A	76	N/A	N/A	50	N/A	N/A

Table 4: The ten top-ranked second-order templates in each domain.

Rank	**Yeast**	**WebKB**	**Twitter**
1	$R_1(X,Y) \vee \neg R_1(Y,X)$	$R_1(X,Y) \vee \neg R_1(Y,X)$	$R_1(X,Y) \vee \neg R_1(X,Z)$
2	$R_1(X,Y) \vee \neg R_1(Y,X) \vee \neg R_2(X,Z)$	$\neg R_1(X,Y) \vee R_1(X,Z) \vee \neg R_1(Y,Z)$	$R_1(X,Y) \vee \neg R_1(Y,X)$
3	$R_1(X,Y) \vee \neg R_1(Z,Y) \vee \neg R_2(X,Z)$	$\neg R_1(X,Y) \vee \neg R_1(Y,X) \vee R_2(X,Z)$	$\neg R_1(X,Y) \vee \neg R_1(Y,X) \vee R_1(Z,X)$
4	$R_1(X,Y) \vee \neg R_1(X,Z) \vee \neg R_1(Y,X)$	$\neg R_1(X,Y) \vee R_1(Y,Z) \vee \neg R_1(Z,X)$	$\neg R_1(X,Y) \vee R_1(X,Z) \vee \neg R_1(Y,X)$
5	$\neg R_1(X,Y) \vee R_1(Y,X) \vee \neg R_2(X,Z)$	$\neg R_1(X,Y) \vee \neg R_1(X,Z) \vee R_1(Y,Z)$	$R_1(X,Y) \vee \neg R_1(X,Z) \vee \neg R_1(Y,Z)$
6	$\neg R_1(X,Y) \vee R_1(X,Z) \vee \neg R_1(Y,Z)$	$R_1(X,Y) \vee \neg R_1(X,Z) \vee \neg R_1(Y,X)$	$\neg R_1(X,Y) \vee R_1(Z,Y) \vee \neg R_2(X,Z)$
7	$\neg R_1(X,Y) \vee R_1(Z,Y) \vee \neg R_2(X,Z)$	$\neg R_1(X,Y) \vee \neg R_1(Z,Y) \vee R_2(X,Z)$	$\neg R_1(X,Y) \vee \neg R_1(Y,X) \vee R_2(X,Z)$
8	$R_1(X,Y) \vee \neg R_1(Y,X) \vee \neg R_1(Z,X)$	$R_1(X,Y) \vee \neg R_1(Z,Y) \vee \neg R_2(X,Z)$	$\neg R_1(X,Y) \vee R_1(Y,X) \vee \neg R_1(Z,X)$
9	$R_1(X,Y) \vee \neg R_1(Y,Z) \vee \neg R_1(Z,X)$	$\neg R_1(X,Y) \vee \neg R_1(X,Z) \vee R_1(Y,X)$	$R_1(X,Y) \vee \neg R_1(Z,Y) \vee \neg R_2(X,Z)$
10	$R_1(X,Y) \vee \neg R_1(X,Z) \vee \neg R_1(Y,Z)$	$R_1(X,Y) \vee \neg R_1(Y,X) \vee \neg R_1(Z,X)$	$R_1(X,Y) \vee \neg R_1(X,Z) \vee \neg R_1(Y,X)$

More extensive results are available in the online supplement.[8] These results contain all the learning curves as well as the AUCPRs and CLLs for all systems.

Conclusion

This paper proposes TODTLER, which is a principled framework for deep transfer learning. TODTLER views knowledge transfer as the process of learning a declarative bias in one domain and transferring it to another to improve the learning process. It applies a two-stage procedure, whereby it learns which second-order patterns are useful in the source domain and biases the learning process in the target domain towards models that have these patterns as well. Our experiments demonstrate that TODTLER outperforms the previous state-of-the-art deep transfer learning approach DTM. While producing more accurate models, TODTLER is also significantly faster than DTM. In the future, we hope to make TODTLER even more powerful by enabling it to transfer a richer set of patterns than any deep transfer learning algorithm can currently handle. In the future, we also want to investigate if TODTLER could be used as a stand-alone MLN structure learning algorithm as Moore and Danyluk (2010) presented some evidence that DTM is well-suited for that task.

Acknowledgments

Jan Van Haaren is supported by the Agency for Innovation by Science and Technology in Flanders (IWT). Jesse Davis is partially supported by the Research Fund KU Leuven (CREA/11/015 and OT/11/051), EU FP7 Marie Curie Career Integration Grant (#294068) and FWO-Vlaanderen (G.0356.12).

References

Banerjee, B.; Liu, Y.; and Youngblood, G. 2006. ICML Workshop on Structural Knowledge Transfer for Machine Learning.

Baxter, J.; Caruana, R.; Mitchell, T.; Pratt, L. Y.; Silver, D. L.; and Thrun, S. 1995. In *NIPS Workshop on Learning to Learn: Knowledge Consolidation and Transfer in Inductive Systems*.

Clauset, A., and Wiegel, F. W. 2010. A Generalized Aggregation-Disintegration Model for the Frequency of Severe Terrorist Attacks. *Journal of Conflict Resolution* 54(1):179–197.

Craven, M., and Slattery, S. 2001. Relational Learning with Statistical Predicate Invention: Better Models for Hypertext. *Machine Learning* 43(1/2):97–119.

Davis, J., and Domingos, P. 2009. Deep Transfer via Second-Order Markov Logic. In *Proceedings of the 26th International Conference on Machine Learning*, 217–224.

Davis, J.; Burnside, E.; Dutra, I.; Page, D.; and Santos Costa, V. 2005. An Integrated Approach to Learning Bayesian Networks of Rules. In *Proceedings of the European Conference on Machine Learning*, 84–95.

Della Pietra, S.; Della Pietra, V.; and Lafferty, J. 1997. In-

[8] Available on http://dtai.cs.kuleuven.be/ml/systems/todtler.

ducing Features of Random Fields. In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, volume 19, 380–392.

Falkenhainer, B.; Forbus, K.; and Gentner, D. 1989. The Structure-Mapping Engine: Algorithm and Examples. *Artificial Intelligence* 41(1):1–63.

Kok, S., and Domingos, P. 2010. Learning Markov Logic Networks Using Structural Motifs. In *Proceedings of the 27th International Conference on Machine Learning*.

Kok, S.; Sumner, M.; Richardson, M.; Singla, P.; Poon, H.; Lowd, D.; Wang, J.; Nath, A.; and Domingos, P. 2010. The Alchemy System for Statistical Relational AI. *Technical Report, Department of Computer Science and Engineering, University of Washington, Seattle, WA. http://alchemy.cs.washington.edu.*

Lowd, D., and Domingos, P. 2007. Efficient Weight Learning for Markov Logic Networks. In *Proceedings of the 11th European Conference on Principles and Practices of Knowledge Discovery in Databases*, 200–211.

Mewes, H. W.; Frishman, D.; Gruber, C.; Geier, B.; Haase, D.; Kaps, A.; Lemcke, K.; Mannhaupt, G.; Pfeiffer, F.; Schüller, C.; Stocker, S.; and Weil, B. 2000. MIPS: A Database for Genomes and Protein Sequences. *Nucleic Acids Research* 28(1):37–40.

Mihalkova, L.; Huynh, T.; and Mooney, R. J. 2007. Mapping and Revising Markov Logic Networks for Transfer Learning. In *Proceedings of the 22nd AAAI Conference on Artificial Intelligence*, 608–614.

Moore, D. A., and Danyluk, A. 2010. Deep Transfer as Structure Learning in Markov Logic Networks. In *Proceedings of the AAAI-2010 Workshop on Statistical Relational AI (StarAI).*

Pan, S. J., and Yang, Q. 2010. A Survey on Transfer Learning. *IEEE Transactions on Knowledge and Data Engineering* 22(10):1345–1359.

Richardson, M., and Domingos, P. 2006. Markov Logic Networks. *Machine Learning* 62(1-2):107–136.