Deep Transfer via Second-Order Markov Logic

Jesse Davis **Pedro Domingos**
Department of Computer Science and Engineering
University of Washington
Seattle, WA 98195-2350, U.S.A.
{*jdavis, pedrod*}@*cs.washington.edu*

Abstract

Standard inductive learning requires that training and test instances come from the same distribution. Transfer learning seeks to remove this restriction. In shallow transfer, test instances are from the same domain, but have a different distribution. In deep transfer, test instances are from a different domain entirely (i.e., described by different predicates). Humans routinely perform deep transfer, but few learning systems, if any, are capable of it. In this paper we propose an approach based on a form of second-order Markov logic. Our algorithm discovers structural regularities in the source domain in the form of Markov logic formulas with predicate variables, and instantiates these formulas with predicates from the target domain. Using this approach, we have successfully transferred learned knowledge between a molecular biology domain and a Web one. The discovered patterns include broadly useful properties of predicates, like symmetry and transitivity, and relations among predicates, like various forms of homophily.

Introduction

Inductive learning has traditionally been defined as generalizing from training instances to test instances from the same distribution. Decades of research have produced many sophisticated techniques for solving this task. Unfortunately, in real applications training and test data often come from different distributions. Humans are able to cope with this much better than machines. In fact, humans are even able to take knowledge learned in one domain and apply it to an entirely different one. For example, Wall Street firms often hire physicists to solve finance problems. Even though these two domains have superficially nothing in common, training as a physicist provides knowledge and skills that are highly applicable in finance (e.g., solving differential equations and performing Monte Carlo simulations). In contrast, a model learned on physics data would simply not be applicable to finance data, because the variables in the two domains are different. What is missing is the ability to discover *structural* regularities that apply to many different domains, irrespective of their superficial descriptions. For example, two domains may be modeled by the same type of equation, and solution techniques learned in one can be applied in the other. The inability to do this is arguably the biggest gap between current learning systems and humans.

Transfer learning addresses this by explicitly assuming that the source and target problems are different. Interest in it has grown rapidly in recent years (e.g., (Driessens, Ramon, & Croonenborghs 2006; Lee *et al.* 2007; Mihalkova, Huynh, & Mooney 2007; Roy & Kaelbling 2007; Taylor & Stone 2007; Torrey *et al.* 2007)). Work to date falls mainly into what may be termed *shallow transfer*: generalizing to different distributions over the same variables, or different variations of the same domain (e.g., different numbers of objects). What remains largely unaddressed is *deep transfer*: generalizing across domains (i.e., between domains where the types of objects and variables are different).

There is a large body of work in a related field: analogical reasoning (e.g., (Falkenhainer, Forbus, & Gentner 1989)). Here knowledge from one domain is applied in another by establishing a correspondence between the objects and relations in them. However, this typically requires human help in establishing the correspondences, and is carried out in an *ad hoc* manner. Further, whatever domain-independent knowledge was used remains implicit. Our goal is to develop a well-founded, fully automatic approach to deep transfer, that makes domain-independent knowledge explicit, modular, and an object of discourse in its own right.

An approach that meets this goal must have a number of properties. It must be relational, since only relational knowledge can be transferred across domains. It must be probabilistic, to handle the uncertainty inherent in transfer in a principled way. Lastly, it must be second-order, since expressing domain-independent knowledge requires predicate variables. These requirements rule out standard inductive logic programming and first-order probabilistic approaches. To our knowledge, the only available representation that satisfies all the criteria is the second-order extension of Markov logic introduced by Kok and Domingos (2007). We use it as the basis for our approach.

Our approach to transfer learning, called DTM (Deep Transfer via Markov logic), discerns high-level similarities between clauses. DTM uses clique templates over sets of literals with predicate variables in place of predicate names. It evaluates all legal clique templates up to a maximum size. DTM evaluates which clique templates represent regularities whose probability differ significantly from the uniform distribution. It selects the top *k* highest scoring clique templates to transfer, and instantiates the knowledge with the types of objects and variables present in the target domain. The transferred knowledge provides a declarative bias for structure learning in the target domain.

The most similar approach to ours in the literature is Mihalkova et al.'s TAMAR algorithm (2007). TAMAR learns Markov logic clauses in the source domain, and attempts to transfer each one to the target domain by replacing its predicates by target-domain predicates in all possible ways. While simple, this approach is unlikely to scale to the large, rich domains where transfer learning is most likely to be useful. Like analogical reasoning, it does not explicitly create domain-independent knowledge.

Using our approach, we have successfully transferred learned knowledge between a molecular biology domain and a Web one. In addition to improved empirical performance, our approach discovered patterns include broadly useful properties of predicates, like symmetry and transitivity, and relations among predicates, such as homophily.

Markov Logic

Formulas in first-order logic consist of four types of symbols: constants, variables, functions, and predicates. Constant symbols represent objects in the domain of interest (e.g., people: `Anna`, `Bob`, `Chris`, etc.). Variable symbols range over the objects in the domain. Function symbols (e.g., `MotherOf`) represent mappings from tuples of objects to objects. Predicate symbols represent relations among objects in the domain (e.g., `Friends`) or attributes of objects (e.g., `Smokes`). A *term* is a variable or a function applied to a tuple of terms. A predicate applied to a tuple of terms is an *atomic formula* or *atom*. A *literal* is an atomic formula or its negation. A *ground atomic formula* or *ground atom* is an atomic formula that contains no variables. A *clause* is a disjunction over a finite set of literals. A *possible world* assigns a truth value to each possible ground predicate.

A *Markov network* is a model for the joint distribution of a set of variables $X = (X_1, X_2, \ldots, X_n)$ (Della Pietra, Della Pietra, & Lafferty 1997). It is composed of an undirected graph G and a set of potential functions ϕ_k. The graph has a node for each variable, and the model has a potential function for each clique in the graph. A potential function is a non-negative real-valued function of the state of the corresponding clique. The joint distribution represented by a Markov network is given by

$$P(X=x) = \frac{1}{Z}\prod_k \phi_k(x_{\{k\}}) \tag{1}$$

where $x_{\{k\}}$ is the state of the kth clique (i.e., the state of the variables that appear in that clique). The *partition function* $Z = \sum_{x\in\mathcal{X}} \prod_k \phi_k(x_{\{k\}})$. Markov networks are often conveniently represented as *log-linear models*, with each clique potential replaced by an exponentiated weighted sum of features of the state, leading to $P(X = x) = \frac{1}{Z}\exp\left(\sum_j w_j f_j(x)\right)$. A feature may be any real-valued function of the state. In the most direct translation from the potential-function form (Eq. 1), there is one feature corresponding to each possible state $x_{\{k\}}$ of each clique, with its weight being $\log \phi_k(x_{\{k\}})$. This representation is exponential in the size of the cliques. However, we are free to specify a much smaller number of features (e.g., logical functions of the state of the clique), allowing for a more compact representation than the potential-function form.

Markov logic is a probabilistic extension of first-order logic. It combines first-order logic with Markov networks. Formally, a Markov logic network (MLN) is a set of pairs, (F_i, w_i), where F_i is a first-order formula and $w_i \in \mathbb{R}$. MLNs soften logic by associating a weight with each formula. Worlds that violate formulas become less likely, but not impossible. Intuitively, as w_i increases, so does the strength of the constraint F_i imposes on the world.

MLNs provide a template for constructing Markov networks. When given a finite set of constants, the formulas from an MLN define a Markov network. Nodes in the network are the ground instances of the preciates in the formulas. Arcs connect ground atoms that appear in the same ground instance of a formula. A MLN induces the following probability distribution: $P(X = x) = \frac{1}{Z}\exp\left(\sum_{i=1}^{F} w_i n_i(x)\right)$, where F is the number formulas in the MLN, w_i is the weight of the i^{th} formula, and $n_i(x)$ is the number of true groundings of F_i in possible world x.

Algorithms have been proposed for learning the weights associated with each formula (Richardson & Domingos 2006) , as well as for learning the formulas themselves (Kok & Domingos 2005). However, structure learning is more intimately tied to transfer learning, so we will discuss it in more depth. Structure learning consists of two components: constructing clauses and evaluating clauses. Clause construction follows an inductive logic programming style search. When learning a network from scratch, it starts with an empty clause and specializes it by successively adding literals to the clause. Additionally, the algorithm can refine an existing network to correct errors in the clauses. In this setting, it is necessary to consider both removing and adding literals to a clause as well as flipping the sign of a literal in the clause. The algorithm uses a beam search to find the current best clause and add it to the network. The search ends when no clause improves the score of the MLN.

To evaluate the merit of each clause it uses weighted pseudo-log-likelihood (WPLL):

$$\log P_w^{\bullet}(X=x) = \sum_{r\in R} c_r \sum_{k=1}^{g_r} \log P_w(X_{r,k}=x_{r,k}|MB_x(X_{r,k})) \tag{2}$$

where R is the set of first-order predicates, g_r is the number of groundings of first-order predicate `r`, $x_{r,k}$ is the truth value (0 or 1) of the kth grounding of `r` and $MB_x(X_l)$ is the state of X_l's *Markov blanket* in the data, and

$$P(X_l=x_l|MB_x(X_l)) = \frac{\exp\left(\sum_{i=1}^{F} w_i n_i(x)\right)}{\exp\left(\sum_{i=1}^{F} w_i n_i(x_{[X_l=0]})\right)+\exp\left(\sum_{i=1}^{F} w_i n_i(x_{[X_l=1]})\right)} \tag{3}$$

where $n_i(x)$ is the number of true groundings of the ith formula in x, $n_i(x_{[X_l=0]})$ is the number of true groundings of the ith formula when we force $X_l = 0$ and leave the remaining data unchanged, and similarly for $n_i(x_{[X_l=1]})$. To avoid overfitting, each clauses receives a penalty term proportional to the number of predicates that differ between the current clause and the initial clause.

Deep Transfer via Markov Logic

The clauses in an MLN capture regularities that hold in the data for a given domain. However, the knowledge that the clauses encode is specific to the types of objects and variables present in that domain. Deep transfer attempts to generalize learned knowledge across domains that have different types of objects and variables. We call our approach Deep Transfer via Markov Logic (DTM). In order to abstract away the superficial domain description, DTM uses second-order Markov logic, where formulas contain predicate variables (Kok & Domingos 2007) to model common structures among first-order clauses. To illustrate the intuition behind DTM, consider the following two clauses

$$\mathtt{Complex(z,y)} \wedge \mathtt{Interacts(x,z)} \Rightarrow \mathtt{Complex(x,y)}$$

$$\mathtt{Location(z,y)} \wedge \mathtt{Interacts(x,z)} \Rightarrow \mathtt{Location(x,y)}$$

These clauses are both instantiations of the following clause:

$$\mathtt{r(z,y)} \wedge \mathtt{s(x,z)} \Rightarrow \mathtt{r(x,y)}$$

where r and s are predicate variables. Introducing predicate variables allows DTM to represent high-level structural regularities in a domain-independent fashion. This knowledge can be transferred to another problem, where the clauses are instantiated with the appropriate predicate names. The key assumption that DTM makes is that the target domain shares some second-order structure with the source domain.

DTM uses clique templates to model second-order structure. It is preferable to use a clique template representation as opposed to clauses because multiple different clauses capture the same regularity. A clique template groups those clauses with similar effects into one structure. A set of literals with predicate variables, such as $\{\mathtt{r(x,y)}, \mathtt{r(y,x)}\}$, defines each clique template. The feature templates for a clique template are conjunctions over the literals in the clique. It is more convenient to look at conjunctions than clauses as they do not overlap, and can be evaluated separately. The feature templates are all possible ways of negating the literals in a clique template. The feature templates for $\{\mathtt{r(x,y)}, \mathtt{r(y,x)}\}$ are: $\{\mathtt{r(x,y)} \wedge \mathtt{r(y,x)}, \mathtt{r(x,y)} \wedge \neg\mathtt{r(y,x)}, \neg\mathtt{r(x,y)} \wedge \neg\mathtt{r(y,x)}\}$.
This template only has three states, as the two ways of negating one literal are equivalent modulo variable renaming.

The three key elements of DTM introduced in the next sections are: (i) how to define legal clique and feature templates, (ii) how to score the cliques templates and (iii) how to apply clique templates to a new problem.

Legal Clique and Feature Templates

For a given domain, DTM evaluates all clique templates subject to the following constraints:

1. The clique template contains either two or three literals, which is done for reasons of tractability.
2. The clique template contains at most three variables, which is done for reasons of tractability.
3. The clique template contains at least one valid instantiation in the source domain.
4. The literals in the clique template are connected. That is, a path of shared variables must connect each pair of literals in the clique template.
5. No two clique templates are the same modulo variable renaming. For example, $\{\mathtt{r(x,y)}, \mathtt{r(z,y)}, \mathtt{s(x,z)}\}$ and $\{\mathtt{r(z,y)}, \mathtt{r(x,y)}, \mathtt{s(z,x)}\}$, where r and s are predicate variables, are equivalent because the second template renames variable x to z and variable z to x.

While this represents a limited set of clique templates, it seems to work well in practice. In the future we will address this limitation by discovering and transferring longer second-order templates.

The feature templates for each clique template are conjunctions over the literals in the clique template. The feature templates are all possible ways of negating the literals in the clique template subject to the following constraints:

1. No two feature templates are the same modulo variable renaming.
2. Two distinct variables are not allowed to unify. For example, $\mathtt{r(x,y)} \wedge \mathtt{r(y,x)}$, really represents the following formula: $\mathtt{r(x,y)} \wedge \mathtt{r(y,x)} \wedge \mathtt{x} \neq \mathtt{y}$. This constraint ensures that a possible grounding does not satisfy feature templates for two separate clique templates. For example, without this constraint, all true groundings of $\mathtt{r(x,y)} \wedge \mathtt{r(z,y)} \wedge \mathtt{s(x,x)}$ would also be true of $\mathtt{r(x,y)} \wedge \mathtt{r(z,y)} \wedge \mathtt{s(x,z)}$ (which appears as feature template in a different clique template).

Clique Template Evaluation

Intuitively, the goal is to assign a high score to a clique template that captures a dependency between its literals. Each template can be decomposed into sub-templates, and it should capture a regularity beyond what its sub-templates represent. For example the following clique template $\{\mathtt{r(x,y)}, \mathtt{s(x,z)}\}$ can be decomposed into subclique templates $\{\mathtt{r(x,y)}\}$ and $\{\mathtt{s(x,z)}\}$. The original clique is a product of these two subcliques if these factors are independent.

To score a clique template, DTM checks if its probability distribution is significantly different than the product of the probabilities of each possible pair of subcliques that it can be decomposed into. To do this, DTM looks at all first-order instantiations of a template. To check whether a clique's probability is significantly different than its decompositions, DTM uses the K-L divergence:

$$D(p||q) = \sum_f p(f) \log \frac{p(f)}{q(f)} \tag{4}$$

where p is the clique's probability distribution, and q is the one predicted by the decomposition, and f represents a feature. For each feature (conjunction) in the clique, DTM computes the probability that the feature holds in the database. The probability of a feature f is $n(f)/n$, where $n(f)$ is the number of true groundings of f in the database, and n is the number of possible groundings of f.

For each instantiation of a clique template, DTM computes its K-L divergence versus all its decompositions. Each instantiation receives the minimum K-L score over the set of its decompositions, because any single one could explain the clique's performance. Each clique template receives the average score of its top three instantiations in order to favor clique templates that have multiple useful instantiations (we picked three a priori and did not tune this parameter).

Transfer Mechanism

Next, the question becomes how to make use of the transferred knowledge in the target domain. A key component of an inductive logic programming (ILP) system is the declarative bias. Due to the large search space of possible first-order clauses, devising a good declarative bias is crucial for achieving good results with an ILP system. In ILP, two primary methods exist for expressing a declarative bias and both forms of bias are often used in the same system. The first method restricts the search space. Common strategies include having type constraints, forcing certain predicate arguments to contain bound variables, and setting a maximum clause length. The second method involves incorporating background knowledge. Background knowledge comes in the form of hand-crafted clauses that define additional predicates which can be added to a clause under construction. Effectively, background knowledge allows the learner to add multiple literals to a clause at once and overcome the myopia of greedy search. It is important to note that these common strategies can easily be expressed in second-order logic, and this is part of what motivates our approach. DTM can be viewed as a way to learn the declarative bias in one domain and apply it in another, as opposed to having a user hand-craft the bias for each domain. We investigate three different ways to reapply the knowledge in the target domain:

Transfer by Seeding the Beam. In the first approach, the second-order clique templates provide a declarative bias for the standard MLN structure search. DTM selects the top k clique templates that have at least one true grounding in the target domain. DTM seeds the beam with each legal instantiation of a clique template (i.e. those that conform with the type constraints of the domain) in the target domain at the beginning of each iteration of the beam search. This strategy forces certain clauses to be evaluated in the search process that would not be scored otherwise and helps overcome some of the limitations of greedy search. Throughout the remainder of the discussion we use clauses instead of conjunctions. In logic a duality exists between clauses and conjunctions. Negating each literal in a clause translates it into a conjunction that is the negation of the clause. In Markov logic this can be implemented by negating each literal in a clause and flipping the sign of the weight associated with the clause. Additionally, most structure learning approaches, both in Markov logic and ILP, construct clauses.

Greedy Transfer without Refinement. The second approach again picks the top k clique templates that have at least one true grounding in the target domain and creates all legal instantiations in the target domain. This algorithm imposes a very stringent bias by performing a structure search where it only considers including the transferred clauses. The algorithm evaluates all clauses and greedily picks the one that leads to the biggest improvement in WPLL. The search terminates when the addition of any clause does not improve the WPLL.

Greedy Transfer with Refinement. The final approach adds a refinement step to the previous algorithm. In this case, the MLN generated by the greedy procedure serves as seed network during standard structure learning. In this case, the MLN search can refine the clauses picked by the greedy procedure to better match the target domain.

Empirical Evaluation

In this section, we evaluate our approach on two real-world data sets. We compare the DTM algorithm to the Markov logic structure learning (MSL) algorithm described in Section 2. MSL is implemented in the publicly available Alchemy package (Kok *et al.* 2008). We made two modifications to MSL. First, when counting the number of true groundings of a clause, we do not permit two distinct variables to unify to the same constant. We did this to ensure that we evaluate clauses in the same manner that we evaluate clique templates. Second, we modify MSL to learn clauses that contain constants in them. We made this modification as we are interested in predicting specific values of attributes, such as the class label of a web page, for each object in a domain. We first describe the domains and characteristics of the data sets we use and then present and discuss our experimental results.

Tasks

The data for the **Yeast Protein** task come from the MIPS (Munich Information Center for Protein Sequence) Comprehensive Yeast Genome Database as of February 2005 (Mewes *et al.* 2000). The data set, originally used in (Davis *et al.* 2005), includes information on protein location, function, phenotype, class, and enzymes. It also includes information about protein-protein interaction and protein complex data.

The data contain information about approximately 4500 proteins that are involved in an interaction. We created four disjoint subsamples of this data that each contain around 450 proteins. To create each subsample, we randomly selected a seed set of proteins. We included all previously unselected proteins that appeared within two links (via the `Interaction` predicate) of the seed set. For this data set, we attempt to predict the truth value of all groundings of two predicates: `Function` and `Interaction`.

The **WebKB** data set consists of labeled web pages from the computer science departments of four universities. We used the relational version of the data set from (Craven & Slattery 2001), which features 4140 web pages and 10,009 web links, and neighborhoods around each link. Each web page is marked with some subset of the following categories: person, student, faculty, professor, department, research project, and course. For this data set, we again try two tasks. First, we perform the "collective classification" task, and predict the class label for each web page. Second, we perform the "link prediction" task by predicting whether a hyperlink exists between each pair of web pages.

High-Scoring Clique Templates

An important evaluation measure for transfer learning is whether it discovers and transfers relevant knowledge. In fact, DTM discovers multiple broadly useful properties. For example, among clique templates of length three, $\{\mathtt{r(x,y)},\mathtt{r(z,y)},\mathtt{s(x,z)}\}$ is ranked first in both domains. This represents the concept of homophily, which is present when related objects (x and z) tend to have similar properties (y). The template $\{\mathtt{r(x,y)},\mathtt{r(y,z)},\mathtt{r(z,x)}\}$ is ranked second in Yeast and fourth in WebKB and represents the concept of transitivity. Both homophily and transitivity are important concepts that appear in a variety of domains. Therefore it is extremely significant that DTM discovers and assigns a high ranking to these concepts. In WebKB, the second- and third-ranked templates are about pairs of features where at least one of the features occurs on a web page that contains a self-referential link. These templates do not appear to be particularly meaningful. In future work we will explore in depth why they receive a high score.

In both domains, the top three clique templates of length two are: $\{\mathtt{r(x,y)},\mathtt{r(z,y)}\}$, $\{\mathtt{r(x,y)},\mathtt{r(x,z)}\}$, and $\{\mathtt{r(x,y)},\mathtt{r(y,x)}\}$. The first template captures the fact that particular feature values are more common across objects in a domain than others. For example, a computer science department most likely has more student web pages than faculty web pages. The second template captures the fact that pairs of values for the same feature, such as words in the WebKB domain, commonly co-occur in the same object. The final template captures symmetry, another important general property of relations.

Methodology

The central question our experiments aimed to address was whether DTM improved performance in the target domain compared to learning from scratch with MSL. As a second baseline, we included the performance achieved by just using the unit clauses, which capture the marginal probability that each predicate is true. This baseline provided a quick check to ensure that we were actually learning something in each domain. Each domain was divided into four disjoint subsets, which we called mega-examples. We used one mega-example to train the learner and then tested on the remaining three mega-examples. We repeated this train-test cycle for all mega-examples.

Within each task, all algorithms had the same parameter settings. Each algorithm was given at most 48 hours to run. In each domain, we optimized the WPLL for the two predicates we were interested in predicting. For DTM we tried two settings for the parameter $k = 5, 10$, which determined the number of clauses transferred to the target domain. In Yeast, we permitted only function constants to appear in learned clauses. We wanted to know exactly which functions (e.g. metabolism, DNA processing, etc.) each protein performed, not just that the protein had some function. In WebKB, we permitted only page class constants to appear in clauses. Here we wanted to predict which labels (e.g. student, person, etc.) applied to each web page, not just whether the web page has a label. We restricted the algorithm to only learn these constants for reasons of tractability. Allowing other types of would have significantly enlarged the search space.

For all approaches, learning the structure required us to perform subsampling on some ground atoms. In Yeast, we subsampled 10,000 false groundings of the `Interaction` predicate and in WebKB we subsampled 10,000 false groundings of the `Linked` predicate. Furthermore, we sampled at most 10,000 groundings for each clause during structure learning. Thus after obtaining the final MLN, we relearned the weights associated with each formula using all of the training data.

To evaluate each system, we measured the test set conditional log-likelihood (CLL) and area under the precision-recall curve (AUC) for each predicate. We determined the CLL of a ground predicate according to Equation 3. The advantage of the CLL is that it directly measures the quality of the probability estimates produced. For a particular decision threshold, recall measures the fraction of positive examples that are correctly labeled and precision measures that fraction of examples classified as positive that are truly positive. The advantage of the AUC is that it is insensitive to the large number of true negatives.

Discussion of Results

Table 1 reports the AUCs and CLLs, averaged over all four folds, for each algorithm on the Yeast data set. DTM outperforms MSL on all metrics except for AUC when predicting the interaction predicate. Each learning method yields better results than only using the unit clauses. The transfer approaches of seeding the beam and refining the greedily constructed network both result in better CLLs and AUCs than MSL when predicting protein function. Additionally, each results in better CLLs than MSL when predicting whether two proteins interacts, apart from the refinement approach that only uses the top five patterns. Greedy and MSL have roughly equivalent performance. The transfer approaches all have a lower AUC than MSL when predicting interaction. This degradation in performance is probably a result of the mismatch between the WPLL (which is correlated with CLL) and AUC.

Only when using the transfer algorithm that refines the greedily learned MLN do we see a large difference between selecting the top five and top ten highest scoring clique templates for transfer. In this case, going from five to ten patterns results in large improvements in AUC and CLL for both predicates. The difference in performance occurs from small changes in two folds. In each case, the greedy algorithm adds clauses from outside the five highest scoring clique templates, which are subsequently refined. In the first case, template that represents transitivity is selected. In the second case, two clauses are added that reflect the fact that very few proteins interact with themselves.

Table 1 reports the AUCs and CLLs, averaged over all four folds, for each algorithm on the WebKB data set. On this data set DTM results in better performance than MSL.

Table 1: Experimental results when transferring the top ten clique templates. AUC stands for average area under the precision-recall curve. CLL stands for average conditional log-likelihood. Unit stands for the MLN with only unit clauses. MSL stands for the standard structure learning algorithm in Alchemy. Seed stands for seeding the beam with the transferred clauses. Greedy stands for greedily adding transferred clauses. Refine stands for refining the greedily learned network.

	Function		Interaction		Page Class		Linked	
Algorithm	AUC	CLL	AUC	CLL	AUC	CLL	AUC	CLL
Unit	0.077	−0.272	0.006	−0.035	0.068	−0.251	0.002	−0.016
MSL	0.237	−0.245	**0.067**	−0.034	0.811	−0.117	0.063	−0.015
Seed	0.292	−0.230	0.046	−0.033	0.786	−0.117	0.0582	−0.015
Greedy	0.243	−0.257	0.056	−0.033	0.834	−0.078	**0.112**	**−0.014**
Refine	**0.413**	**−0.197**	0.049	**−0.033**	**0.844**	**−0.068**	0.085	**−0.014**

Each learning method results in better performance than only using the unit clauses. The transfer approaches of greedily learning the MLN and refining the greedily constructed network each result in better CLLs and AUCs for both predicates than the other methods. The greedy approach has a slightly higher AUC when predicting the presence of a link between web pages whereas the refinement algorithm has a better CLL. This is a result of the previously mentioned mismatch between CLL and AUC. Seeding the beam results in lower AUCs than MSL but with slightly better CLLs. The small number of predicates and types in this domain may make the stronger bias imposed by the greedy transfer algorithm less restraining. Here no difference exists between transferring the top five and top ten highest scoring clique templates. Yeast ranks the more interesting (from a human interpretability perspective) in the top five and empirically these are the most useful clique templates.

Conclusions and Future Work

DTM successfully transferred learned knowledge between a molecular biology domain and a Web one. Empirically, knowledge learned in the molecular biology domain improved our ability to assign class labels to web pages and predict whether one page contains a hyperlink that points to a second page. Additionally, knowledge gleaned from the Web domain enhanced our ability to predict a protein's function. Finally, DTM discovered explicit, domain-independent knowledge that included broadly useful properties of predicates, like symmetry and transitivity, and relations among predicates, like various forms of homophily. Currently DTM is restricted to work with a small set of clique templates and not all relevant concepts can be captured. In the future we will address this limitation by discovering and transferring longer second-order clique templates.

Acknowledgments

We would like to thank Inês Dutra, Daniel Lowd, Vítor Santos Costa and Marc Sumner for reading over drafts of this paper. This research was partly funded by DARPA contracts NBCHD030010 02-000225, FA8750-07-D-0185, and HR0011-07-C-0060, DARPA grant FA8750-05-2-0283, NSF grant IIS-0534881, and ONR grant N-00014-05-1-0313. The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of DARPA, NSF, ONR, or the United States Government.

References

Craven, M., and Slattery, S. 2001. Relational learning with statistical predicate invention: Better models for hypertext. *Machine Learning* 43.

Davis, J.; Burnside, E.; Dutra, I.; Page, D.; and Costa, V. S. 2005. An integrated approach to learning Bayesian networks of rules. In *Proc. ECML'05*, 84–95.

Della Pietra, S.; Della Pietra, V.; and Lafferty, J. 1997. Inducing features of random fields. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 19:380–392.

Driessens, K.; Ramon, J.; and Croonenborghs, T. 2006. Transfer learning for reinforcement learning through goal and policy parametrization. In *Proc.ICML'06 Workshop on Structural Knowledge Transfer for Machine Learning*.

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

Kok, S., and Domingos, P. 2005. Learning the structure of Markov Logic Networks. In *Proc. ICML'05*, 441–448.

Kok, S., and Domingos, P. 2007. Statistical predicate invention. In *Proc. ICML'07*, 433–440.

Kok, S.; Sumner, M.; Richardson, M.; Singla, P.; Poon, H.; Lowd, D.; and Domingos, P. 2008. 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.

Lee, S.-I.; Chatalbashev, V.; Vickrey, D.; and Koller, D. 2007. Learning a meta-level prior for feature relevance from multiple related tasks. In *Proc. ICML'07*, 489–496.

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 *Proc. AAAI'07*, 608–614.

Richardson, M., and Domingos, P. 2006. Markov logic networks. *Machine Learning* 62:107–136.

Roy, D. M., and Kaelbling, L. P. 2007. Efficient bayesian task-level transfer learning. In *Proc. IJCAI'07*, 2599–2604.

Taylor, M. E., and Stone, P. 2007. Cross-domain transfer for reinforcement learning. In *Proc. ICML'07*, 879–886.

Torrey, L.; Walker, T.; Shavlik, J.; and Maclin, R. 2007. Relational macros for transfer in reinforcement learning. In *Proc. ILP'07*.