Deep Transfer via Second-Order Markov Logic

Jesse Davis JDAVIS@CS.WASHINGTON.EDU
Pedro Domingos PEDROD@CS.WASHINGTON.EDU
Department of Computer Science & Engineering, University of Washington, Seattle, WA 98195, USA

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 among molecular biology, social network and Web domains. The discovered patterns include broadly useful properties of predicates, like symmetry and transitivity, and relations among predicates, such as various forms of homophily.

1. 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., Baxter et al., 1995; Silver et al., 2005; Banerjee et al., 2006; Taylor et al., 2008). 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 et al., 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.

Appearing in *Proceedings of the 26th International Conference on Machine Learning*, Montreal, Canada, 2009. Copyright 2009 by the author(s)/owner(s).

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 logical formulas. Given a set of first-order formulas, DTM lifts the formulas into second-order logic by replacing all predicate names with predicate variables. It then groups the second-order formulas into cliques. DTM evaluates which second-order cliques represent regularities whose probability deviates significantly from independence among subcliques. Finally, it selects the top k highest scoring second-order cliques 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 among social network, molecular biology and web domains. In addition to improved empirical performance, our approach discovered patterns including broadly useful properties of predicates, like symmetry and transitivity, and relations among predicates, such as homophily.

2. Markov Logic

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 et al., 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. The joint distribution represented by a Markov network is: $P(X=x) = \frac{1}{Z}\prod_k \phi_k(x_{\{k\}})$, where $x_{\{k\}}$ is the state of the kth clique (i.e., the state of the variables that appear in that clique), and Z is a normalization constant. 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: $P(X = x) = \frac{1}{Z}\exp\left(\sum_j w_j f_j(x)\right)$. A feature $f_j(x)$ may be any real-valued function of the state.

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. MLNs provide a template for constructing Markov networks. When given a finite set of constants, the formulas from an MLN define a Markov network with one node for each ground predicate and one feature for each ground clause. The weight of a feature is the weight of the first-order clause that originated it. An 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 (e.g., Lowd & Domingos, 2007), as well as for learning the formulas themselves (e.g., Kok & Domingos, 2005). We will focus on the algorithm of Kok and Domingos, which we will call MSL. Structure learning consists of two components: constructing clauses and evaluating clauses. Clause construction follows an inductive logic programming-style search (Dzeroski & Lavrac, 2001). When learning a network from scratch, MSL 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. Here, it considers 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). To avoid overfitting, each clause receives a penalty term proportional to the number of literals that differ between the current clause and the initial clause.

3. Deep Transfer via Markov Logic

The formulas in an MLN capture regularities that hold in the data for a given domain. However, the knowledge that the formulas encode is specific to the types of objects and predicates present in that domain. Deep transfer attempts to generalize learned knowledge across domains that have different types of objects and predicates. 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 formulas. To illustrate the intuition behind DTM, consider the formulas $\texttt{Complex}(\texttt{z},\texttt{y}) \wedge \texttt{Interacts}(\texttt{x},\texttt{z}) \Rightarrow \texttt{Complex}(\texttt{x},\texttt{y})$ and $\texttt{Location}(\texttt{z},\texttt{y}) \wedge \texttt{Interacts}(\texttt{x},\texttt{z}) \Rightarrow \texttt{Location}(\texttt{x},\texttt{y})$. Both are instantiations of: $\texttt{r}(\texttt{z},\texttt{y}) \wedge \texttt{s}(\texttt{x},\texttt{z}) \Rightarrow \texttt{r}(\texttt{x},\texttt{y})$, where $\texttt{r}$ and $\texttt{s}$ 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 formulas 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.

Given a set of first-order formulas, DTM converts each formula into second-order logic by replacing all predicate names with predicate variables. It then groups the second-order formulas into cliques. Two second-order formulas are assigned to the same clique if and only if they are over the same set of literals. DTM evaluates which second-order cliques represent regularities whose probability deviates significantly from independence among their subcliques. It selects the top k highest-scoring second-order cliques to transfer to the target domain. The transferred knowledge provides a declarative bias for structure learning in the target domain.

The four key elements of DTM, introduced in the next subsections, are: (i) how to define cliques, (ii) how to search for cliques, (iii) how to score the cliques and (iv) how to apply cliques to a new problem.

3.1. Second-order Cliques

DTM uses second-order cliques to model second-order structure. It is preferable to use this representation as opposed to arbitrary second-order formulas because multiple different formulas over the same predicates can capture the same regularity. A clique groups those formulas with similar effects into one structure. A set of literals with predicate variables, such as $\{\texttt{r}(\texttt{x},\texttt{y}), \texttt{r}(\texttt{y},\texttt{x})\}$, defines each second-order clique and the states, or features, 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 features correspond to all possible ways of negating the literals in a clique. The features for $\{\texttt{r}(\texttt{x},\texttt{y}), \texttt{r}(\texttt{y},\texttt{x})\}$ are $\{\texttt{r}(\texttt{x},\texttt{y}) \wedge \texttt{r}(\texttt{y},\texttt{x})\}, \{\texttt{r}(\texttt{x},\texttt{y}) \wedge \neg\texttt{r}(\texttt{y},\texttt{x})\}, \{\neg\texttt{r}(\texttt{x},\texttt{y}) \wedge \texttt{r}(\texttt{y},\texttt{x})\}$ and $\{\neg\texttt{r}(\texttt{x},\texttt{y}) \wedge \neg\texttt{r}(\texttt{y},\texttt{x})\}$. Relational Markov networks (RMNs) (Taskar et al., 2002) are another representation language that makes use of relational cliques. However, RMNs only look at first-order relations, and are a special case of Markov logic, whereas DTM makes use of second-order logic. Furthermore, RMNs do not allow uncertainty over relations, only over attributes, and they scale poorly with clique size.

DTM imposes the following restrictions on cliques:

1. The literals in the clique are connected. That is, a path of shared variables must connect each pair of literals in the clique.
2. No cliques are the same modulo variable renaming. For example, $\{\texttt{r}(\texttt{x},\texttt{y}), \texttt{r}(\texttt{z},\texttt{y}), \texttt{s}(\texttt{x},\texttt{z})\}$ and $\{\texttt{r}(\texttt{z},\texttt{y}), \texttt{r}(\texttt{x},\texttt{y}), \texttt{s}(\texttt{z},\texttt{x})\}$, where $\texttt{r}$ and $\texttt{s}$ are predicate variables, are equivalent as the second clique renames variable $\texttt{x}$ to $\texttt{z}$ and variable $\texttt{z}$ to $\texttt{x}$.

The states of a clique are all possible ways of negating the literals in the clique subject to the following constraints:

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

3.2. Search

DTM works with any learner than induces formulas in first-order logic. This paper evaluates two separate strategies for inducing formulas in the source domain: exhaustive search and beam search.

Exhaustive search. Given a source domain, the learner generates all first-order clauses up to a maximum clause length and a maximum number of object variables. The entire set of clauses is passed to DTM for evaluation.

Beam search. Exhaustive search does not scale well, since the number of clauses it produces is exponential in the clause length and it becomes computationally infeasible to score long clauses. Beam search is a common strategy for scaling structure learning, and is used in MSL (described in Section 2). However, transfer learning and structure learning have different objectives. In transfer learning, the goal is to derive a large, diverse set of clauses to evaluate for potential transfer to the target domain. Structure learning simply needs to induce a compact theory that accurately models the predicates in the source domain. The theories induced by MSL tend to contain very few clauses and thus are not ideal for transfer. An alternative approach is to induce a separate theory to predict each predicate in the domain. However, the resulting theory may not be very diverse, since clauses will contain only the target predicate and predicates in its Markov blanket (i.e., its neighbors in the network). A better approach is to construct models that predict sets of predicates.

Given the final set of learned models, DTM groups the clauses into second-order cliques and evaluates each clique that appears more than twice.

3.3. Second-Order Clique Evaluation

Each clique can be decomposed into pairs of subcliques, and it should capture a regularity beyond what its subcliques represent. For example, the second-order clique $\{\mathtt{r(x,y)}, \mathtt{r(z,y)}, \mathtt{s(x,z)}\}$ can be decomposed into the following three pairs of subcliques: (i) $\{\mathtt{r(x,y)}, \mathtt{r(z,y)}\}$ and $\{\mathtt{s(x,z)}\}$, (ii) $\{\mathtt{r(x,y)}, \mathtt{s(x,z)}\}$ and $\{\mathtt{r(z,y)}\}$, and (iii) $\{\mathtt{r(z,y)}, \mathtt{s(x,z)}\}$ and $\{\mathtt{r(x,y)}\}$.

To score a second-order clique, each of its first-order instantiations is evaluated. To score a first-order clique, DTM checks if its probability distribution is significantly different from the product of the probabilities of each possible pair of subcliques that it can be decomposed into.

The natural way to compare these two distributions is to use the K-L divergence, $D(p||q) = \sum_x p(x) \log \frac{p(x)}{q(x)}$, where p is the clique's probability distribution, and q is the distribution it would have if the two subcliques were independent. We use Bayesian estimates of the probabilities with Dirichlet priors with all $\alpha_i = 1$.

For each first-order instantiation of a second-order clique, 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 distribution. Each second-order clique receives the average score of its top m first-order instantiations, in order to favor second-order cliques that have multiple useful instantiations.

3.4. Transfer Mechanism

The next question is how to make use of the transferred knowledge in the target domain. A key component of an inductive logic programming (ILP) (Dzeroski & Lavrac, 2001) 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.

When applying a second-order clique in a target domain, DTM decomposes the clique into a set of clauses, and transfers each clause. It uses clauses instead of conjunctions since most structure learning approaches, both in Markov logic and ILP, construct clauses. In Markov logic, a conjunction can be converted into an equivalent clause by negating each literal in it and flipping the sign of its weight.

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 cliques provide a declarative bias for the standard MLN structure search. DTM selects the top k cliques that have at least one true grounding in the target domain. At the beginning of each iteration of the beam search, DTM seeds the beam with the clauses obtained from each legal instantiation of a clique in the target domain (i.e., only instantiations that conform with the type constraints of the domain are considered). 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.

Greedy Transfer without Refinement. The second approach again picks the top k second-order cliques 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 no clause addition improves 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. The MLN search can now refine the clauses picked by the greedy procedure to better match the target domain. Additionally, it can induce new clauses to add to the theory.

DTM differs from previous approaches to declarative bias in two main ways: it is probabilistic, and it provides the bias entirely automatically. For example, relational clichés (Silverstein & Pazzani, 1991) are second-order formulas that define potential refinements for a candidate clause. However, the clichés are all hand-coded. DTM is also more fully automated than the Clint/Cia approach (De Raedt & Bruynooghe, 1992), which uses an oracle to determine which background knowledge predicates to construct.

A more automated approach is that of Bridewell and Todorovski (2007), which uses ILP to learn the declarative bias for process modeling domains. Their algorithm analyzes a series of models in order to discern the components that are common among accurate models, which then become the bias for new domains. This approach requires a set of previously learned models in order to derive the bias, while DTM only needs data from one source domain. Additionally, this approach would require further hand-crafting of background knowledge in order to extend it beyond process modeling problems.

4. Empirical Evaluation

In this section, we evaluate our approach on three real-world data sets. We compare the DTM algorithm to the Markov logic structure learning (MSL) algorithm described in Section 2 and to TAMAR. MSL is implemented in the publicly available Alchemy package (Kok et al., 2009). 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 cliques. Second, we modify MSL to allow learning clauses that contain constants. We made this modification since 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.

4.1. Domains

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; Davis et al., 2005). The data set 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 some 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 and 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.

The **Social Network** data set consists of pages collected from the Facebook social networking Web site. The data consist of information about friendships among individuals and properties of individuals, such as hobbies, interests and network memberships. It has information about approximately 600 individuals, thirteen different properties of each and 24,000 friendship pairs.

4.2. High-Scoring Second-order Cliques

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 cliques of length three with up to three object variables, $\{\mathtt{r(x,y)}, \mathtt{r(z,y)}, \mathtt{s(x,z)}\}$ is ranked first in all three domains. This represents the concept of homophily, which is present when related objects (x and z) tend to have similar properties (y). The clique $\{\mathtt{r(x,y)}, \mathtt{r(y,z)}, \mathtt{r(z,x)}\}$ is ranked second in Yeast and fourth in WebKB and Facebook and represents the concept of transitivity. Both homophily and transitivity are important concepts that appear in a variety of domains. Therefore it is quite significant that DTM discovers and assigns a high ranking to these concepts.

In all three domains, the top three cliques 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 clique 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 clique 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 clique captures symmetry, another important general property of relations.

4.3. 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. We tried four source-target pairs: Facebook to Yeast Protein, WebKB to Yeast Protein, Facebook to WebKB and Yeast Protein to WebKB. Each target domain was divided into four disjoint subsets, which we called mega-examples. We selected a subset of the mega-examples to train the learner on and then tested it on the remaining mega-examples. We repeated this train-test cycle for all possible subsets of the mega-examples.

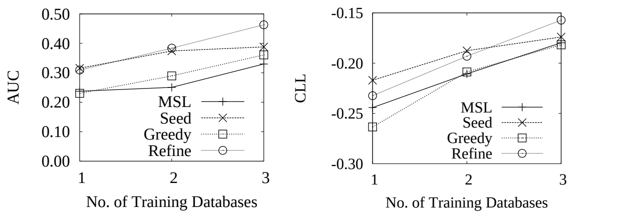
Within each domain, all algorithms had the same parameter settings. Each algorithm was allotted 100 hours per database to run. In each domain, we optimized the WPLL for the two predicates we were interested in predicting. For DTM we tried two settings, $k = 5$ and $k = 10$ for the number k of second-order cliques transferred to the target domain. We tried two settings for exhaustive search. The first evaluated all clauses containing at most three literals and three object variables and the second evaluated all clauses containing up to four literals and four object variables. When running beam search in the source domain, we constructed theories for predicting pairs of predicates (a tradeoff between diversity and tractability).

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 page class 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. For each train-test split, we used the information gain (on the training set) to pick the top fifty words most predictive of page class. For tractability, we restricted the algorithm to only learn with these constants.

To evaluate each system, we measured the test set conditional log-likelihood (CLL) and area under the precision-recall curve (AUC) for each predicate. The advantage of the CLL is that it directly measures the quality of the probability estimates produced. The advantage of the AUC is that it is insensitive to the large number of true negatives.

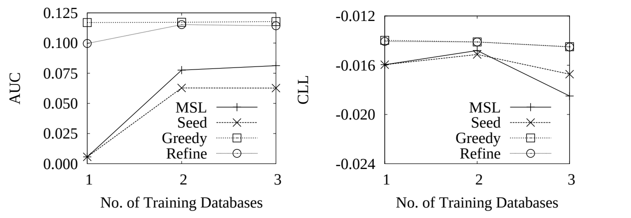
4.4. Results

Figure 1 shows representative learning curves for predicting protein function when transferring cliques of up to length four from the WebKB domain into the Yeast domain. DTM consistently outperforms MSL. As expected, DTM performs better relative to MSL with less data. Table 1 compares the performance, measured by average relative difference (e.g., $(AUC_{DTM} - AUC_{MSL})/AUC_{MSL}$) of DTM using greedy transfer with refinement to MSL when transferring cliques of up to length three. In this domain, transfer greatly improves the models' ability to predict both the AUC and CLL for the `Function` predicate. For this predicate, in the length three setting, DTM outperforms MSL on 73% (82 out of 112) of the various different conditions. In the length four setting, it wins 75% of the cases (42 out of 56) when using WebKB as the



(a) AUC (b) CLL

Figure 1. Learning curves for the `Function` predicate when transferring second-order cliques of length four from the WebKB domain.



(a) AUC (b) CLL

Figure 2. Learning curves for the `Linked` predicate when transferring second-order cliques of length four from the Facebook domain.

source domain. Using Facebook in this setting did not provide a benefit. The cliques transferred here tend to capture more information about link structure and the various properties of a single entity, whereas in the Yeast domain it is important to model similarities and differences in the properties of related entities. Neither approach does well at the difficult task of predicting the Interaction predicate; MSL outperforms DTM.

Figure 2 shows representative learning curves for predicting whether two Web pages are linked when transferring cliques of up to length four from Facebook into WebKB. As in Figure 1, DTM consistently outperforms MSL. Table 2 compares the performance of DTM using greedy transfer with refinement to MSL for WebKB when transferring cliques of up to length three. Across both predicates, transfer outperforms MSL on 68% (152 out of 224) of the various different test conditions when considering cliques of up to length three. When using Facebook as the source task, transferring cliques of up to length four provides a large benefit, winning 71% (79 out of 112) of time. When using Yeast as the source domain, transfer only posts modest gains, winning only 51% (58 out of 112) of the various test conditions. On the `Linked` predicate, the benefit of transfer is greater for small amounts of data. Transfer also provides a slight improvement over MSL when predicting the label of a Web page.

Due to space limitations, we omit the tables and graphs of the results for beam search[1]. When transferring from the WebKB domain into the Yeast domain, the results are almost identical to exhaustive search. For the Facebook into Yeast pair, the results are worse than exhaustive search. When transferring from the Yeast domain into the WebKB domain, the results are comparable to exhaustive search. For the Facebook into WebKB pair, the `PageClass` results were slightly better than exhaustive search, but worse for the `Linked` predicate. In general, exhaustive search seems to yield better and more stable results than beam search. One possible explanation is that despite tweaking the beam search algorithm to make it more appropriate for transfer, it still focuses too heavily on finding a good theory as opposed to finding the best clauses for transfer.

We also compared our approach to TAMAR (Mihalkova et al., 2007). In order to have a fair comparison, we extended the system to map clauses that contain constants in them. When transferring from Yeast to WebKB, TAMAR does significantly worse than all other approaches. TAMAR was unable to map theories from WebKB into the Yeast domain in the allotted time. Introducing constants into the clauses greatly increases the number of possible mappings between clauses, making it infeasible to perform an exhaustive search to map clauses between different domains. This problem is particularly acute when mapping into the Yeast domain, due to the number of function constants.

5. Conclusion

This paper proposes an approach to deep transfer, the problem of generalizing across domains. Our DTM algorithm identifies domain-independent regularities in the form of second-order Markov logic clauses, and uses them to guide discovery of new structure in the target domain. Initial experiments in bioinformatics, Web and social network domains show that DTM outperforms standard structure learning, and discovers significant regularities like homophily and transitivity.

Directions for future work include: theoretical analysis of DTM; improved structure learning algorithms for transfer; transferring formulas instead of whole cliques; experiments in richer domains; transferring from multiple domains at once; etc.

[1]Full results are available in an online appendix: http://alchemy.cs.washington.edu/papers/davis09.

Table 1. Experimental results comparing DTM (greedy transfer with refinement) to MSL on the Yeast domain. Each entry in the table is the average relative difference in AUC or CLL between transfer and MSL when considering second-order cliques that contain up to three predicate and object variables.

Source	Num. Cliques	AUC						CLL					
		Function			Interaction			Function			Interaction		
		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	5	0.48	0.70	0.30	-0.31	-0.33	-0.21	0.12	0.17	0.03	0.01	0.03	-0.02
FB	5	0.29	0.13	-0.22	0.00	0.56	-0.26	0.02	-0.10	-0.23	0.00	0.06	-0.14
WebKB	10	0.47	0.52	0.21	-0.18	-0.32	-0.07	0.12	0.10	0.05	0.04	0.02	0.01
FB	10	0.63	0.51	0.10	-0.35	-0.03	-0.42	0.18	0.01	-0.16	0.01	0.07	-0.01

Table 2. Experimental results comparing DTM (greedy transfer with refinement) to MSL on the WebKB domain. Each entry in the table is the average relative difference in AUC or CLL between transfer and MSL when considering second-order cliques that contain up to three predicate and object variables.

Source	Num. Cliques	AUC						CLL					
		Page Class			Linked			Page Class			Linked		
		1 DB	2 DB	3 DB	1 DB	2 DB	3 DB	1 DB	2 DB	3 DB	1 DB	2 DB	3 DB
Yeast	5	-0.01	0.02	0.00	42.79	9.77	11.03	-0.12	0.04	0.03	0.12	0.04	0.19
FB	5	0.00	0.01	0.01	40.79	4.64	6.15	-0.06	0.02	0.07	0.12	0.04	0.19
Yeast	10	-0.01	0.02	-0.01	42.79	10.36	10.94	-0.12	0.03	0.00	0.12	0.04	0.19
FB	10	0.00	0.01	0.01	40.79	4.64	6.15	-0.06	0.02	0.07	0.12	0.04	0.19

Acknowledgements

This research was partly funded by ARO grant W911NF-08-1-0242, DARPA contracts FA8750-05-2-0283, FA8750-07-D-0185, HR0011-06-C-0025, HR0011-07-C-0060 and NBCH-D030010, NSF grants IIS-0534881 and IIS-0803481, and ONR grant N00014-08-1-0670. 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 ARO, DARPA, NSF, ONR, or the United States Government.

References

Banerjee, B., Liu, Y., & Youngblood, G. (Eds.). (2006). *ICML workshop on structural knowledge transfer for machine learning.*

Baxter, J., Caruana, R., Mitchell, T., Pratt, L. Y., Silver, D. L., & Thrun, S. (Eds.). (1995). *NIPS workshop on learning to learn: Knowledge consolidation and transfer in inductive systems.*

Bridewell, W., & Todorovski, L. (2007). Learning declarative bias. *Proc. ILP'07* (pp. 63 – 77).

Craven, M., & Slattery, S. (2001). Relational learning with statistical predicate invention: Better models for hypertext. *Machine Learning*, *43*, 97–119.

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

De Raedt, L., & Bruynooghe, M. (1992). Interactive concept-learning and constructive induction by analogy. *Machine Learning*, *8*, 107–150.

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

Dzeroski, S., & Lavrac, N. (Eds.). (2001). *Relational data mining.* Berlin, Germany: Springer.

Falkenhainer, B., Forbus, K. D., & Gentner, D. (1989). The structure-mapping engine: Algorithm and examples. *Artificial Intelligence*, *41*, 1–63.

Kok, S., & Domingos, P. (2005). Learning the structure of Markov logic networks. *Proc. ICML'05* (pp. 441–448).

Kok, S., & Domingos, P. (2007). Statistical predicate invention. *Proc. ICML'07* (pp. 433–440).

Kok, S., Sumner, M., Richardson, M., Singla, P., Poon, H., Lowd, D., Wang, J., & Domingos, P. (2009). *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., & Domingos, P. (2007). Efficient weight learning for Markov logic networks. *Proc. PKDD'07* (pp. 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., & Weil, B. (2000). MIPS: a database for genomes and protein sequences. *Nucleic Acids Research*, *28*, 37–40.

Mihalkova, L., Huynh, T., & Mooney, R. J. (2007). Mapping and revising Markov logic networks for transfer learning. *Proc. AAAI'07* (pp. 608–614).

Silver, D., Bakir, G., Bennett, K., Caruana, R., Pontil, M., Russell, S., & Tadepalli, P. (Eds.). (2005). *NIPS workshop on inductive transfer: 10 years later.*

Silverstein, G., & Pazzani, M. J. (1991). Relational clichés: Constraining constructive induction during relational learning. *Proc. ICML'91* (pp. 203–207).

Taskar, B., Abbeel, P., & Koller, D. (2002). Discriminative probabilistic models for relational data. *Proc. UAI'02* (pp. 485–492).

Taylor, M., Fern, A., & Driessens, K. (Eds.). (2008). *AAAI workshop on transfer learning for complex tasks.*