Mapping across relational domains for transfer learning with word embeddings-based similarity*

Thais Luca[1][0000−0001−5999−5901], Aline Paes[2][0000−0002−9089−7303], and Gerson Zaverucha[1][0000−0002−3641−6839]

[1] Department of Systems Engineering and Computer Science, Universidade Federal do Rio de Janeiro, Rio de Janeiro, RJ, Brazil
{tluca,gerson}@cos.ufrj.br
[2] Institute of Computing, Universidade Federal Fluminense, Niteroi, RJ, Brazil
alinepaes@ic.uff.br

Abstract. Statistical machine learning models are a concise representation of probabilistic dependencies among the attributes of an object. Most of the models assume that training and testing data come from the same distribution. Transfer learning has emerged as an essential technique to handle scenarios where such an assumption does not hold, as it relies on leveraging the knowledge acquired in one or more learning tasks as a starting point to solve a new task. Statistical Relational Learning (SRL) extends statistical learning to represent and learn from data with several objects and their relations. In this way, SRL deals with data with a rich vocabulary composed of classes, objects, their properties, and relationships. When employing transfer learning to SRL, the primary challenge is to transfer the learned structure, mapping the vocabulary from a source domain to a different target domain. To address the problem of transferring across domains, we propose TransBoostler, which uses pre-trained word embeddings to guide the mapping as the name of a predicate usually has a semantic connotation that can be mapped to a vector space model. After transferring, TransBoostler employs theory revision operators further to adapt the mapped model to the target data. In the experimental results, TransBoostler has successfully transferred trees from a source to a distinct target domain, performing equal or better than previous work but requiring less training time.

Keywords: Transfer learning · Statistical relational learning · Word embeddings.

1 Introduction

In traditional machine learning methods, data is represented in a tabular format. This type of representation despises the relational structure of the data, which contains crucial information about how objects participate in relationships and events. Most of the real-world data is relational, and exploring the structure of relational data allows finding solutions to problems of higher complexity [9].

* Supported by CAPES, FAPERJ, and CNPq.

Statistical Relational Learning (SRL) combines elements from statistical and relational modeling to relational learning aiming at representing and learning in domains with complex relational and rich probabilistic structure [11]. SRL has succeeded in many real-world applications as real data also require handling uncertainty from noise and incomplete information. Most of SRL models assume training and testing data must belong to the same feature space and have the same distribution. If those distributions differ from each other, a new model must be trained using newly collected data.

To address the existence of training and testing data from different distributions, transfer learning [33] has emerged as an important technique given that collecting new data can be a costly or even impossible task. Transfer learning has recently gained much interest from researchers due to its success in Deep Learning applications [27]. It relies on leveraging the knowledge acquired in one or more learning tasks to achieve a good initial performance for solving a new task. Furthermore, they also target at reducing the amount of time it takes to learn a model from scratch [24]. More importantly, applying transfer learning to SRL models admits training and testing domains to differ in distributions as it has successfully been verified in previous works [1, 13, 16]. However, relational learning differs from function-based and traditional machine learning methods since the former data has a rich vocabulary composed of classes, objects, their properties, and relationships [10]. Therefore, the challenge of applying transfer learning to SRL models is primarily how to transfer the learned structure, mapping the vocabulary from a source domain to the most appropriate objects, properties, and relations in a different target domain. TreeBoostler [1], a system that employs transfer learning to a Relational Dependency Boosting (RDN-B) framework, recursively tries to transfer nodes from source regression trees to build target regression trees. It tries every possible mapping of a source predicate and chooses the best mapping using weighted variance as the decision criterion. This approach can be costly and time-consuming. Thus, devising other more efficient mechanisms for mapping the vocabulary is vital.

In this work, we propose to use pre-trained word embeddings to guide the mapping as the name of the predicates usually have a semantic connotation that can be mapped to a Vector Space Model (VSM). The mechanism proposed here is named as TransBoostler[3]. As TreeBoostler, it also focuses on transferring Boosted Relational Dependency Networks (RDNs) but uses pre-trained word vector representations of predicates for mapping. It maps the predicates that appear in trees learned in the source domain to the most similar predicates in a target domain. Thus, there is no searching as it takes advantage of the context of embeddings to choose the best mapping. As TreeBoostler, our algorithm also includes a Theory Revision component to propose modifications in order to count on predicates from the target domain that were not mapped to any predicate from the source domain. Also, the revision component may accommodate modifications pointed out by the target training data.

[3] The source code and experiments are publicy available at https://github.com/MeLL-UFF/TransBoostler.

We evaluated TransBoostler in real-world relational datasets training on one single fold and testing on the remaining folds to simulate the scenario of few data available. We tested mapping by similarity using three different similarity metrics. Our results demonstrate that the proposed algorithm can successfully transfer learned theories across different domains by mapping predicates using similarity. However, mapping by similarity can impair performance depending on the source and target domains.

This paper is organized as follows. Section 2 introduces necessary background. Section 3 compares our proposal to other algorithms in the literature. Section 4 presents TransBoostler and how it performs predicate mapping. Section 5 describes experimental results of TransBoostler for different datasets. Finally, Section 6 presents our conclusions.

2 Background Knowledge

This section presents an overview of concepts used to build the contributions of this paper, namely, functional gradient boosting to learn RDNs, transfer learning, word embeddings, and the similarity metrics we relied on to map predicates.

2.1 Functional gradient boosting of relational dependency networks

In real-world applications, the logical structure of the data may contain crucial information about how objects interact with each other. This information about how objects participate in relationships and events can help reach conclusions about other objects allowing for solving even more complex problems [9]. Relational datasets store data across multiple tables, where each table represents different types of entities and how these entities may relate to each other.

Following first-order logic (FOL), relations are represented as logical facts while domains are represented using constants, variables and predicates. For instance, the logical fact *publication("Title", "Jane Doe")* can be used to represent the relation between a published material identified by *Title* and the person named as *Jane Doe* that wrote that material. In this case, *publication* is the name of the predicate while *Title* and *Jane Doe* are constants representing the entities of the domain. Arguments may be associated with a type. In our example, the first argument is associated with the type *title* while the second is associated with the type *person*. Since we have two arguments, we say this predicate is of arity two. Predicates can also represent properties of objects. Two examples are *actor* and *director* to distinguish if a person is an actor or a director. Both have one single argument of type *person*, so their arity is one. An atom is a predicate applied to terms. A term can be a variable, constant or function symbol applied to terms. Atoms that assert a relationship among constants are called ground atoms (e.g. *publication(title, person)*). A literal can be an atom or a negated atom. A disjunction of literals with only one positive literal is a definite clause.

Statistical Relational Learning (SRL) combines elements from statistical and probabilistic modeling to represent and learn in domains with complex relational

and rich probabilistic structure [11]. SRL models are a concise representation of probabilistic dependencies among the attributes of different related objects. One of such models are the Relational Dependency Networks (RDNs) [23]. RDNs are graphical models that have the capacity of expressing and reasoning dependencies. They approximate the joint distribution of a set of random variables as a product of conditional distributions over a ground atom. RDNs consist of a set of *predicates* and *function* symbols where associated with each predicate Y_i in the domain is a conditional probability distribution $P(Y_i|\mathbf{Pa}(Y_i))$ that defines the distribution over the values of Y_i given its parents' values $\mathbf{Pa}(Y_i)$ [23].

RDN-B [22] proposes to represent each conditional probability distribution in a relational dependency network as a weighted sum of regression models. These models grow in a stage-wise optimization and, instead of representing the conditional distribution for each attribute (relation) as a single relational probability tree, they build a set of relational regression trees using gradient-based boosting. As boosting is a nonparametric approach, the number of parameters grows with the number of training episodes. Thus, interactions between random variables are introduced as needed and the large search space is not explicitly considered. RDN-B is fast and straightforward to implement. Most importantly, it enables learning structure and parameters simultaneously.

2.2 Transfer Learning

Transfer learning has emerged primarily to handle scenarios where training and testing distributions differ. Besides, as it leverages previous learned knowledge to address a new domain, it may reduce the need for huge amounts of data. As data can be easily outdated, and newly data can be expensive or impossible to collect, it may be the key to reduce re-calibration effort as a model trained in one time period can be adapted to predict data in a new time period [24, 33]. Transfer learning aims at providing machine learning methods with the ability of recognizing knowledge previously learned in a source domain and apply this knowledge to a new model in a target domain. It contributes to improving performance and tends to make learning a new task less time- and data consuming, as exploiting knowledge learned from one or more previous tasks avoids learning from scratch one specific domain. For SRL models, the main challenge is how to transfer vocabulary from a source domain into a quite different target domain.

2.3 Word Embeddings

Pre-trained continuous word representations have become a great asset for Natural Language Processing (NLP) and machine learning applications [26]. Word embeddings are the most common and useful way to represent words as dense vectors of fixed length.

When it comes to word embeddings, a word is characterized by the company it keeps, such that words that appear in similar contexts must have similar meanings [6]. For instance, *student* and *professor* tend to have similar semantics since they usually appear in similar contexts. The term *embedding* is commonly used

to refer to word representations generated using neural networks as the standard approach to learning these representations is to train log-bilinear models as Word2Vec [19] and fast-Text [4].

Fast-Text, the model used in this work, aims at predicting the words in the surrounding context given a target word. It relies on negative sampling and embeddings created from sub-words to tackle out-of-vocabulary words. Thus, fast-Text considers each word is represented as a bag of character n-gram. The problem of predicting context words is modeled as a set of independent binary classification tasks, where the goal is to predict the presence or absence of context words. Given a word at position t, called w_t, all context words are considered as positive examples and negative samples are chosen at random from a dictionary. For each word w in the vocabulary there are two vectors, $\mathbf{u}_{w_t}$ and $\mathbf{v}_{w_c}$, corresponding to the target word w_t and the same word acting as context word w_c, respectively. Given a dictionary of n-grams of size G, the n-grams appearing in w can be denoted by $G_w \subset \{1, \ldots, G\}$ and a vector representation $\mathbf{z}_g$ is associated to each n-gram g. Each word is represented by the sum of its n-grams representations. The score function is the sum of the dot product between $\mathbf{z}_g$ and word vector $\mathbf{v}_{w_c}$ for every n-gram g in G_w. In this way, fast-Text allows sharing the representation across words and does not ignore its internal structure.

Word embeddings are useful for operations like distance measures [26]. Here, we apply three distance measures to find a mapping between a pair of predicates:

Euclidean Distance is the simplest way to measure distance between two real-valued vectors. The distance between two vectors $\mathbf{p}, \mathbf{q} \in \mathbb{R}^n$ is given by 1.

$$d(p, q) = ||p - q||_2 \tag{1}$$

Soft Cosine Measure is a modification of the traditional cosine similarity measure as it takes into account the similarity between features (words) in the VSM [25]. It considers the cosine similarity of each pair of features to build a matrix of similarity $s_{i,j}$ which introduces new features to the VSM. Thus, the Soft Cosine similarity between two vectors $\mathbf{p}, \mathbf{q} \in \mathbb{R}^n$ is given by Equation 2. If there is no similarity between features, $s_{ii} = 1$ and $s_{ij} = 0$ for $i \neq j$, is equivalent to the traditional cosine similarity measure.

$$soft\_cosine(p, q) = \frac{\sum\sum_{i,j}^{N} s_{i,j} p_i q_j}{\sqrt{\sum\sum_{i,j}^{N} s_{i,j} p_i p_j} \sqrt{\sum\sum_{i,j}^{N} s_{i,j} q_i q_j}} \tag{2}$$

Word Mover's Distance (WMD) also considers the semantic similarity between word pairs [14]. It considers the "travel cost" between words to obtain the minimum cumulative cost of moving a given document $\mathbf{d}$ to a document $\mathbf{d}'$. Assuming text documents are represented as normalized bag-of-words (nBOW) vectors, the similarity between individual word pairs is given by the Euclidean distance and WMD solves the linear transportation problem to obtain how costly it is to travel from one document to another.

3 Related Work

The main challenge of applying transfer learning to learn relational data is how to transfer vocabulary. LTL [13] performs type-matching to identify predicates in the target domain that are similar in their relational structure to predicates in the source domain. It performs type-based tree construction and uses theory refinement in each clause by adding or deleting predicates to try to improve its accuracy. The TAMAR [16] algorithm uses weighted pseudo- log-likelihood (WPLL) to transfer Markov Logic Networks from a source domain to a target domain. The mapping that gives the best WPLL in the target domain is used as mapping for a clause in the source domain. It also revises the structure to improve its accuracy. There are many applications of embeddings for relational tasks. TransE [5] uses embeddings of entities to represent relationships as translations in the vector space of entities and relations. Based on TransE, TransH [31] represents relations as translating operations on a hyperplane, and entities can have multiple representations accordingly to a relation. In [30], authors show evidence that embedded representations can be useful for problems poor in domain knowledge, but results depend on the embedding method used.

TreeBoostler [1] is the most closely related to our algorithm. It performs an exhaustive search by building type constraints to find adequate mappings for a source predicate in the target domain. It uses weighted variance as the decision criterion to choose the best adequate mapping. Then, to improve its accuracy, it revises those trees by pruning and expanding nodes. Our algorithm proposes a modification to TreeBoostler's mapping component. It takes advantage of the semantics of pre-trained word vectors to find mappings by similarity, followed by refining the clauses to better fit the target domain.

4 TransBoostler

Our proposed algorithm comprises the same top-level components as proposed by [1], namely, first the source boosted trees structure are transferred to a different target domain, and next the algorithm revises its trees by pruning and expanding nodes to try to better fit the new data. The main difference is how our algorithm finds adequate predicate mappings.

4.1 Transferring and revising the structure

As TreeBoostler, our algorithm also adopts the local mapping approach introduced by [16], which constructs the best mapping for predicates that appear in each source clause, separately and independently of how other clauses were mapped. Local mapping is generally more scalable since the number of predicates in a single clause is smaller than the total number of predicates of the source domain. It is also more flexible as mapping one part of the structure does not hold or depends on other parts.

Each path from the root to the leaf in a relational regression tree can be seen as a clause in a logic program. These paths may share the same inner nodes with different paths in the tree, so paths are not independent. As RDN-B works on a set of relational trees, these are not independent and cannot be interpreted individually. Thus, the algorithm translates just the predicates that no longer have appeared in the structure. If a predicate has already appeared in the structure, the same mapping is used.

Our proposed algorithm also uses Theory Revision [32] to repair possible faults that can prevent theories from predicting examples correctly. Only mapping the predicates is usually not enough as knowledge comes from a different distribution domain [16]. The theory revision component searches for points in the trees to adjust the initial mapped source theory to fit the target data, hence, improving its inferential capabilities.

The revision component searches for paths in trees that are responsible for bad predictions of examples and defines them as revision points. If a positive example is not covered by the theory, it means the theory is too specific and needs to be generalized, so we call it a specialization point. If theory covers negative examples, it is too general and needs to be specialized, so we call it a generalization point. These points need to be modified in order to increase accuracy, so modifications are proposed to revision points by applying revision operators. The pruning operator (generalization operator) increases coverage of examples by deleting nodes from a tree, while the expansion operator (specialization operator) decreases coverage of examples by expanding nodes in each tree. The Pruning operator is applied first to prune the tree from the bottom to the top recursively, removing nodes whose children are leaves marked as revision points. Then, the Expansion operator is applied and recursively adds nodes that give the best split in a leaf marked as a revision point.

The pruning procedure could prune an entire tree and, if this happens, the revision algorithm would have to expand nodes from an empty tree, which is the same as learning from scratch. If pruning results in a null model, deletion of all the trees, the operator is ignored as if it was never applied. Finally, the revision algorithm uses the conditional log-likelihood (CLL) to score both transferred and revised theory. If the revised theory scores better than before, it is implemented.

4.2 Mapping Component

Assuming predicates have meaningful names, TransBoostler takes advantage of the semantic of pre-trained word vectors to find a suitable mapping by similarity. TransBoostler first builds a list of pairs of predicates ordered by their similarity, computed with metrics over the embeddings of predicates. Next, it follows such an ordered list to employ the most similar mappings. Source trees are pre-trained using RDN-B, and transference starts from the root node of the first source tree and works recursively to find the best mapping for not-mapped predicates.

Text Normalization: In relational datasets, predicates can be made of one, two, or more words. Then, the first step of the transfer process is to split each predicate into its component words. We use Ekphrasis [2] with Wikipedia

corpora for word segmentation. As an example, suppose we have a predicate *athleteplaysforteam*. It should be segmented into *athlete plays for team*. Also, some predicates or words can appear in shortened form, which is the case of "*ta*" that stands for *teaching assistant* and "*tempadvisedby*" that stands for *temporarily advised by*. After word segmentation, every shortened word is replaced by its full form using a pre-built dictionary. Thus, "*ta*" becomes "*teaching assistant*" and "*tempadvisedby*" becomes "*temporarily advised by*". Lastly, WordNet lemmatizer [21] was used to turn verbs into their base form. Verbs are identified in predicates by a Part-Of-Speech Tagger (POS Tagger) tool [28].

Word-Vectors Representation: Word vectors pre-trained on Wikipedia with fast-Text Skip-gram [20] are used to represent each predicate in the VSM. The use of pre-trained word vectors is important as relational datasets have a limited vocabulary. Also, pre-trained word vectors contribute to finding more similar predicates as similar words are approximated by their contexts. If a word does not belong to the pre-trained model vocabulary, it is represented as a null vector. When applying Euclidean distance, word vectors of the same predicate are concatenated to become one single vector. We choose concatenation because it avoids information loss since we have very short sentences. Suppose two predicates *company has office* and *company ceo*. As *company has office* is a 3-dimensional vector with components $(\mathbf{x}, \mathbf{y}, \mathbf{z})$ and *company ceo* is 2-dimensional with components $(\mathbf{x}, \mathbf{w})$, we must express both in the same feature space before concatenating. Then, we create two new vectors whose components represent terms in predicates. To express *company has office*, we set its components to $(\mathbf{x}, \mathbf{y}, \mathbf{z}, \mathbf{0})$. To express *company ceo*, we set its components to $(\mathbf{x}, \mathbf{0}, \mathbf{0}, \mathbf{w})$.

Mapping by Similarity: A source predicate is mapped to a target if they have the highest similarity value in comparison with other targets. Suppose a source i and a list of predicates of the target domain $[x, y, z]$. Then, i is mapped to y if and only if $sim(i, y) > sim(i, x)$ and $sim(i, y) > sim(i, z)$, where sim is the similarity function. We only map predicates of same arity and, if there is a tie, we use alphabetical order. Every source is given the most similar predicate in the target domain. It is not permitted to have more than one distinct source predicate mapped to the same target predicate. If the algorithm cannot find a compatible mapping, the predicate is mapped to "empty".

After transfer, we can have three different scenarios as modeled by [1]: (1) the best scenario is when all literals in an inner node have a non-empty predicate mapping, which means we have the same number of literals in the transferred tree; (2) an inner node has some predicate mapped to "empty", but at least one mapped to non-empty. In this case, the ones mapped to empty are discarded; (3) an inner node that has all its literals mapped to an empty predicate, which is the worst case. Discarding all the literals results in an empty node, which affects the tree structure. Then, the algorithm discards the empty node, promotes its left child and appends its right child to the right-most path of the subtree. If the left child is a leaf, the right child is promoted. If both are leaves, it is discarded.

5 Experimental Results

In this section, we present the experiments performed to evaluate TransBoostler. We compare our results with RDN-B [22], which learns from the target dataset from scratch, and TreeBoostler [1], which is a SRL transfer learning approach.

Experiments Setup For the trees learned by TransBoostler, we follow the same experimental setup as TreeBoostler, so we set the depth limit of trees to be 3, the number of leaves to be 8, the number of regression trees to 10, the maximum number of literals per node to 2, subsampling of negative examples is in a ratio of 2 negatives for 1 positive, and the initial potential is -1.8. Testing is done with all the negative examples. TransBoostler is evaluated with three similarity metrics: Soft Cosine, Euclidean distance, and WMD. We consider two versions of our algorithm: one only considering predicate mapping and parameter learning (TransBoostler*) and the completed version that additionally performs theory revision (TransBoostler). We compare the two versions to evaluate how revising can improve transferring. As TreeBoostler was successfully compared with other transfer methods, these results are omitted.

We used six publicly available datasets paired as in previous literature [1, 13, 16, 18]: (1) IMDB dataset [17] aims at predicting which actor has worked for a director by learning the *workedunder* relation. It is divided into five mega-examples [16] where each one presents information about four movies; (2) Cora [3] is a dataset of Computer Science research papers. It contains 1295 distinct citations to 122 papers and its goal is to predict if two venues represent the same conference by learning the *samevenue* relation. This dataset is divided into five mega-examples; (3) UW-CSE [12] contains information about the Department of Computer Science and Engineering at the University of Washington (UW-CSE). It consists of five mega-examples to predict if a student is advised by a professor; (4) Yeast protein [15] is a dataset obtained from MIPS Comprehensive Yeast Genome Database and contains information about proteins. The goal is to predict if a protein is associated with a class. It consists of four folds independent of each other; (5) Twitter [29] dataset consists of tweets about Belgian soccer matches divided into two independent folds. The goal is to predict the account type (club, fan, or news); and (6) NELL [7] is a machine learning system that extracts information from web texts and converts it into a probabilistic knowledge base. We consider Sports and Finances domains. Sports contains information about athletes, teams, leagues, etc. The goal is to predict which sport is played by a team. Finances contains information about economic sectors of companies, companies' CEOs, companies' country, etc. The goal is to predict if a company belongs to an economic sector. It is split randomly into three folds.

Results Results are averaged over n runs, where, for each run, a new learned source model is used for transference. We used conditional log-likelihood (CLL), area under the ROC curve (AUC ROC), area under the PR curve (AUC PR) [8] and training time as measures to compare performance. We did not consider the

time required to load the fast-Text model, as it is negligible, but we did consider time to calculate similarities between predicates.

The first experiment simulates the transfer learning scenario by learning from a reduced set of data. Following the previous literature, training is performed on one fold and testing on the remaining n-1 folds. Tables 1 and 2 present the transfer experiments for pairs IMDB and Cora, and Yeast and Twitter. Each of them was treated as source and target domains on each turn. Table 3 presents the results for IMDB $\rightarrow$ UW-CSE and NELL Sports $\rightarrow$ NELL Finances. We omitted the opposite transferring from UW-CSE to IMDB and NELL Finances to NELL Sports. The former is too easy, and the latter results in learning from scratch for TreeBoostler as it cannot find useful mappings. We measured the statistical significance between TransBoostler and the baselines using a paired t-test with $p \leq 0.05$. In tables, $\star$ indicates when TransBoostler is significantly better than TreeBoostler. $\diamond$ indicates that the difference between TransBoostler against RDN-B results is significantly better.

As can been seen from the results, TransBoostler performs comparably to baselines in all but one experiment for AUC ROC. Using pre-trained word vectors to find mappings by similarity did improve runtime for transferring pairs Yeast and Twitter. Yeast is one of the largest datasets, and TransBoostler uses far less runtime than TreeBoostler to learn it. To reduce the searching space, TreeBoostler creates type constraints during mapping, which result in some of the predicates being mapped to null. However, TransBoostler maps all source predicates, as it focuses only on similarity. Then, mapping more predicates to a target reduces revision time. As shown in Table 1, for IMDB $\rightarrow$ Cora, our algorithm is more time-consuming. Theories learned using IMDB contain three distinct predicates. Two of them have arity one, and one is of arity two. As Cora has no predicates of arity one, only the predicate of arity two is mapped. Then, it takes more time to revise the structure. TreeBoostler finds *venue* as the best mapping for *movie*, while TransBoostler using Soft Cosine, Euclidean distance and WMD, finds *haswordauthor*, *author*, and *sameauthor*, respectively. For the opposite experiment, Cora $\rightarrow$ IMDB, TransBoostler is competitive to RDN-B and finds the same mappings as TreeBoostler: *haswordvenue* is mapped to *movie*, and *haswordtitle* is mapped to *genre*. This shows mapping by similarity is cogent. As can be seen in Table 3, for NELL Sports $\rightarrow$ NELL Finances, our algorithm underperforms TreeBoostler for AUC ROC but outperforms RDN-B. In this case, TreeBoostler finds four adequate mappings and TransBoostler maps every source predicate to different targets, except when using Euclidean distance.

We also compare the performance of TransBoostler for different amounts of target data. We employed traditional cross-validation methodology as training is performed on n-1 folds and testing on the remaining one. In this experiment, we incrementally increase the amount of target data. Training data is shuffled and divided into five sequence parts. As in the previous experiment, the process is done in n runs, and curves obtained by averaging the results. Experiments are presented in Figures 1, 2, 3, 4, 5, and 6. As can been seen, TransBoostler outperforms or equates the baselines for most experiments, and results show it

Table 1: Comparison between TransBoostler and baselines for IMDB and Cora datasets.

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

Table 2: Comparison between TransBoostler and baselines for Yeast and Twitter datasets.

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

Table 3: Comparison between TransBoostler and baselines for pairs of datasets IMDB and UW-CSE and NELL Sports and NELL Finances.

	IMDB → UW-CSE				NELL Sports → NELL Finances			
	CLL	AUC ROC	AUC PR	Run-time(s)	CLL	AUC ROC	AUC PR	Run-time(s)
RDN-B	-0.257	0.940	0.282	8.74	-0.323	0.692	0.062	24.86
TreeBoostler	-0.247	0.939	0.302	4.78	-0.165	0.980	0.071	124.59
TransBoostler Soft Cosine	-0.255	0.936	0.284	5.94	-0.321⋆	0.721	0.087	62.70
TransBoostler Euclidean	-0.254	0.936	0.275	6.44	-0.320⋆	0.750	0.069	54.04
TransBoostler WMD	-0.247	0.936	0.274	5.89	-0.324⋆⋄	0.741⋄	0.079	53.81
TreeBoostler*	-0.267	0.930	0.293	0.63	-0.315	0.979	0.068	8.85
TransBoostler* Soft Cosine	-0.385⋆⋄	0.608	0.035	1.17	-0.366⋆⋄	0.531	0.001	6.92
TransBoostler* Euclidean	-0.296⋆⋄	0.906	0.131	1.53	-0.365⋆⋄	0.558	0.002	5.88
TransBoostlter* WMD	-0.288⋆⋄	0.906	0.131	1.19	-0.365⋆⋄	0.540	0.002	5.66

performs better than TreeBoostler for smaller amounts of data. The exceptions are AUC ROC and AUC PR curves presented in Figure 1 and AUC ROC presented in Figure 6. These correspond to experiments IMDB → Cora and NELL Sports → NELL Finances, in which TransBoostler has difficulty finding best mappings. Figure 4 also shows Twitter → Yeast underperforms for AUC ROC.

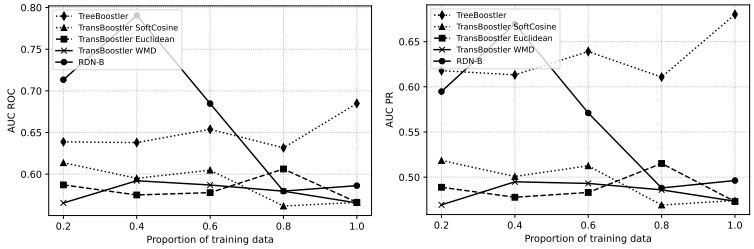

Fig. 1: Learning curves for AUC ROC (left) and AUC PR (right) for IMDB → Cora transfer experiment.

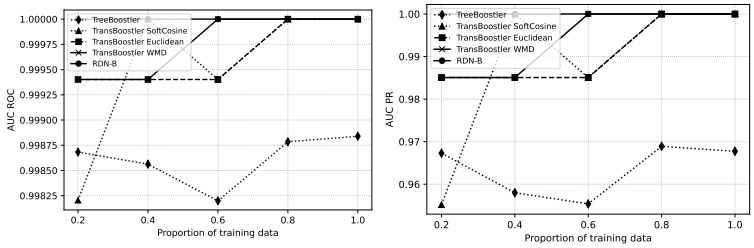

Fig. 2: Learning curves for AUC ROC (left) and AUC PR (right) for Cora → IMDB transfer experiment.

6 Conclusions

In this paper, we presented TransBoostler, an algorithm that transfers Boosted RDNs learned from a source domain to a different target domain. TransBoostler leverages pre-trained word embeddings to find mappings between predicates by similarity. It also relies on Theory Revision to propose modifications to the mapped model by pruning and expanding nodes in order to improve its accuracy. As observed in the experimental results, modifying the mapping component to map by similarity has good performance and can be less time-consuming than a previous related transfer learning approach, depending on the pair of datasets.

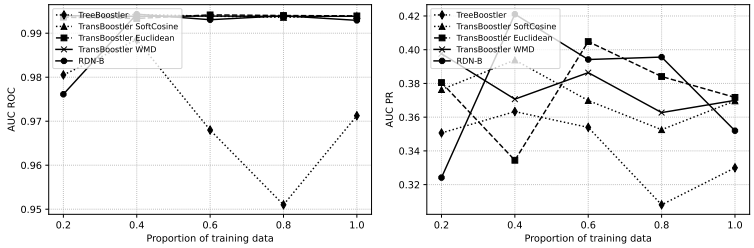

Fig. 3: Learning curves for AUC ROC (left) and AUC PR (right) for Yeast → Twitter transfer experiment.

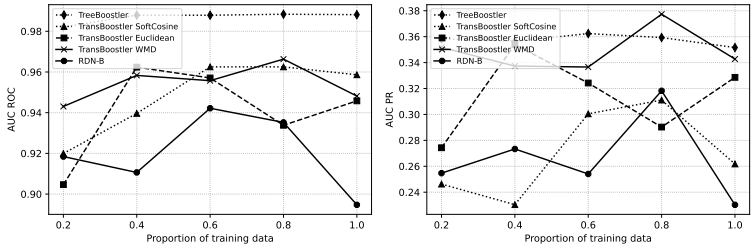

Fig. 4: Learning curves for AUC ROC (left) and AUC PR (right) for Twitter → Yeast transfer experiment.

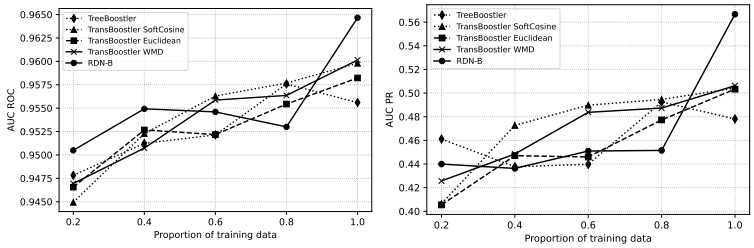

Fig. 5: Learning curves for AUC ROC (left) and AUC PR (right) for IMDB → UW-CSE transfer experiment.

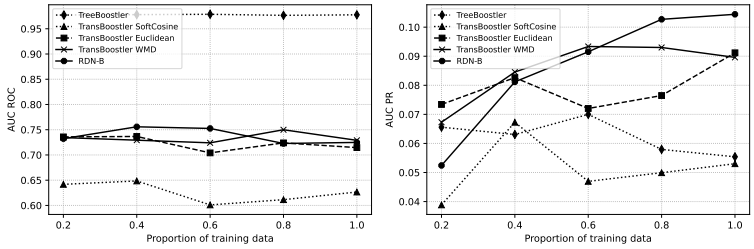

Fig. 6: Learning curves for AUC ROC (left) and AUC PR (right) for NELL Sports → NELL Finances transfer experiment.

It remains a future investigation to understand whether or not to transfer from one domain to another and the effect of the data in which the embeddings were trained. Other possible future is to select the top-N most similar predicates to proceed with the mapping. The proposed mapping component can also be applied to different and more general relational models and testing this component to them is also an interesting future work direction.

References

1. Azevedo Santos, R., Paes, A., Zaverucha, G.: Transfer learning by mapping and revising boosted relational dependency networks. Machine Learning **109**, 1435–1463 (2020)
2. Baziotis, C., Pelekis, N., Doulkeridis, C.: Datastories at semeval-2017 task 4: Deep lstm with attention for message-level and topic-based sentiment analysis. In: Proc. of the 11th Int. Workshop on Semantic Evaluation (SemEval-2017). pp. 747–754. Association for Computational Linguistics, Vancouver, Canada (August 2017)
3. Bilenko, M., Mooney, R.J.: Adaptive duplicate detection using learnable string similarity measures. In: Proc. of the 9th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining. p. 39–48. KDD '03, ACM, New York, NY, USA (2003)
4. Bojanowski, P., Grave, E., Joulin, A., Mikolov, T.: Enriching word vectors with subword information. Transactions of the Association for Computational Linguistics **5**, 135–146 (2017)
5. Bordes, A., Usunier, N., Garcia-Duran, A., Weston, J., Yakhnenko, O.: Translating embeddings for modeling multi-relational data. In: Neural Information Processing Systems (NIPS). pp. 1–9 (2013)
6. Camacho-Collados, J., Pilehvar, M.T.: Embeddings in natural language processing. In: Proc. of the 28th Int. Conf. on Computational Linguistics: Tutorial Abstracts. pp. 10–15 (2020)
7. Carlson, A., Betteridge, J., Kisiel, B., Settles, B., Hruschka Jr, E.R., Mitchell, T.M.: Toward an architecture for never-ending language learning. In: Aaai. vol. 5. Atlanta (2010)
8. Davis, J., Goadrich, M.: The relationship between precision-recall and roc curves. In: Proc. of the 23rd int. Conf. on Machine learning. pp. 233–240 (2006)
9. De Raedt, L.: Logical and relational learning. Springer Science & Business Media (2008)
10. Friedman, N., Getoor, L., Koller, D., Pfeffer, A.: Learning probabilistic relational models. In: IJCAI. vol. 99, pp. 1300–1309 (1999)
11. Getoor, L., Taskar, B.: Statistical relational learning (2007)
12. Khosravi, H., Schulte, O., Hu, J., Gao, T.: Learning compact markov logic networks with decision trees. Machine learning **89**(3), 257–277 (2012)
13. Kumaraswamy, R., Odom, P., Kersting, K., Leake, D., Natarajan, S.: Transfer learning via relational type matching. In: 2015 IEEE International Conference on Data Mining. pp. 811–816. IEEE (2015)
14. Kusner, M., Sun, Y., Kolkin, N., Weinberger, K.: From word embeddings to document distances. In: Bach, F., Blei, D. (eds.) Proc. of the 32nd Int. Conf. on Machine Learning. Proc. of Machine Learning Research, vol. 37, pp. 957–966. PMLR, Lille, France (07–09 Jul 2015), http://proceedings.mlr.press/v37/kusnerb15.html
15. Mewes, H.W., Frishman, D., Güldener, U., Mannhaupt, G., Mayer, K., Mokrejs, M., Morgenstern, B., Münsterkötter, M., Rudd, S., Weil, B.: Mips: a database for genomes and protein sequences. Nucleic acids research **30**(1), 31–34 (2002)

16. Mihalkova, L., Huynh, T., Mooney, R.J.: Mapping and revising markov logic networks for transfer learning. In: Aaai. vol. 7, pp. 608–614 (2007)
17. Mihalkova, L., Mooney, R.J.: Bottom-up learning of markov logic network structure. In: Proc. of the 24th Int. Conf. on Machine Learning. p. 625–632. ICML '07, ACM, New York, NY, USA (2007)
18. Mihalkova, L., Mooney, R.J.: Transfer learning from minimal target data by mapping across relational domains. In: 21st Int. Joint Conf. on Artificial Intelligence. Citeseer (2009)
19. Mikolov, T., Chen, K., Corrado, G., Dean, J.: Efficient estimation of word representations in vector space. arXiv preprint arXiv:1301.3781 (2013)
20. Mikolov, T., Grave, E., Bojanowski, P., Puhrsch, C., Joulin, A.: Advances in pre-training distributed word representations. In: Proc. of the Int. Conf. on Language Resources and Evaluation (LREC 2018) (2018)
21. Miller, G.A.: Wordnet: a lexical database for english. Communications of the ACM **38**(11), 39–41 (1995)
22. Natarajan, S., Khot, T., Kersting, K., Gutmann, B., Shavlik, J.: Gradient-based boosting for statistical relational learning: The relational dependency network case. Machine Learning **86**(1), 25–56 (2012)
23. Neville, J., Jensen, D.: Relational dependency networks. Journal of Machine Learning Research **8**(3) (2007)
24. Pan, S.J., Yang, Q.: A survey on transfer learning. IEEE Transactions on knowledge and data engineering **22**(10), 1345–1359 (2009)
25. Sidorov, G., Gelbukh, A., Gomez Adorno, H., Pinto, D.: Soft similarity and soft cosine measure: Similarity of features in vector space model. Computación y Sistemas **18** (09 2014)
26. Torregrossa, F., Allesiardo, R., Claveau, V., Kooli, N., Gravier, G.: A survey on training and evaluation of word embeddings. International Journal of Data Science and Analytics pp. 1–19 (2021)
27. Torrey, L., Shavlik, J.: Transfer learning. In: Handbook of research on machine learning applications and trends: algorithms, methods, and techniques, pp. 242–264. IGI global (2010)
28. Toutanova, K., Klein, D., Manning, C.D., Singer, Y.: Feature-rich part-of-speech tagging with a cyclic dependency network. In: Proc. of the 2003 Conf. of the North American Chapter of the Association for Computational Linguistics on Human Language Technology - Volume 1. p. 173–180. NAACL '03, ACl, USA (2003)
29. Van Haaren, J., Kolobov, A., Davis, J.: Todtler: Two-order-deep transfer learning. In: Proc. of the 29th AAAI Conference on Artificial Intelligence. vol. 4, pp. 3007–3015. AAAI (2015)
30. Vig, L., Srinivasan, A., Bain, M., Verma, A.: An investigation into the role of domain-knowledge on the use of embeddings. In: Int. Conf. on Inductive Logic Programming. pp. 169–183. Springer (2017)
31. Wang, Z., Zhang, J., Feng, J., Chen, Z.: Knowledge graph embedding by translating on hyperplanes. In: Proc. of the AAAI Conf. on Artificial Intelligence. vol. 28 (2014)
32. Wrobel, S.: First order theory refinement. Advances in inductive logic programming **32**, 14–33 (1996)
33. Yang, Q., Zhang, Y., Dai, W., Pan, S.J.: Transfer Learning. Cambridge University Press (2020)