Explainable Models via Compression of Tree Ensembles

Siwen Yan[1*], Sriraam Natarajan[1], Saket Joshi[2], Roni Khardon[3] and Prasad Tadepalli[4]

[1*]Department of Computer Science, University of Texas at Dallas, 800 W Campbell Rd, Richardson, 75080, TX, USA.
[2]Amazon, USA.
[3]Department of Computer Science, Indiana University, Luddy Hall, 700 N. Woodlawn Avenue, Bloomington, 47408, IN, USA.
[4]Department of Computer Science, Oregon State University, 1500 SW Jefferson Way, Corvallis, 97331, OR, USA.

*Corresponding author(s). E-mail(s): Siwen.Yan@utdallas.edu;
Contributing authors: Sriraam.Natarajan@utdallas.edu; saketjoshi@gmail.com; rkhardon@iu.edu; tadepall@engr.orst.edu;

Abstract

Ensemble models (bagging and gradient-boosting) of relational decision trees have proved to be one of the most effective learning methods in the area of probabilistic logic models (PLMs). While effective, they lose one of the most important aspect of PLMs – interpretability. In this paper we consider the problem of compressing a large set of learned trees into a single explainable model. To this effect, we propose CoTE – Compression of Tree Ensembles – that produces a single small decision list as a compressed representation. CoTE first converts the trees to decision lists and then performs the combination and compression with the aid of the original training set. An experimental evaluation demonstrates the effectiveness of CoTE in several benchmark relational data sets.

Keywords: Tree Combination, Ensemble, TILDE, First Order Logic

1 Introduction

Probabilistic relational logic models (PLMs) provide distributions over logical representations and have been very successful in probabilistic modeling (De Raedt, Kersting, Natarajan, & Poole, 2016; Natarajan, Khot, Kersting, & Shavlik, 2015). In contrast to other successful representations like Neural Networks, a logical representation lends itself to easy interpretation, which is an important factor for the acceptance of ML based solutions in spaces of high accountability (e.g. critical medicine). This advantage of logical representations breaks down under ensemble models like bagged or boosted logical regression trees, because ensembles are not logical models themselves and have more complex semantics. This is somewhat unfortunate because ensembles often improve performance over single models (Natarajan, Khot, Kersting, Gutmann, & Shavlik, 2012). We investigate an approach to translate the ensemble of logical trees into an easily interpretable single relational decision list.

Previous approaches to convert complex models into simpler ones have largely relied on using the complex model to label its own training data and subsequently train the simple model from it (Craven & Shavlik, 1995). This approach has evolved into knowledge distillation (Hinton, Vinyals, & Dean, 2015), a popular technique in the Neural Network community. With knowledge distillation, however, the focus is on reducing model complexity, not improving interpretability. Furthermore, any model-agnostic method for model compression is unable to exploit model specific features, e.g., the logical semantics of the component trees in an ensemble. In contrast, our approach compresses an ensemble of logical trees into a logical decision list (semantically equivalent to a logical decision tree) and avoids re-training the model. A decision list can be considered as an ordered list of first-order logic rules. When evaluating a decision list, the first rule satisfied by an instance is used to generate the prediction or probability for the instance.

Another line of work (Vidal & Schiffer, 2020) recently showed how tree ensembles can be combined into a single tree without re-training. Their approach, however, is strictly for propositional representations. In contrast, we allow and exploit rich relational structure in our model.

Our incremental compression approach called *Compression of Tree Ensembles* (CoTEs) is based on the following key ideas:

1. We exploit the similarity of the semantics of relational trees and decision lists which allows us to soundly translate single tree into a decision list without introducing complicating factors such as negations and existential quantification.
2. We compose trees incrementally interspersed by subsumption inference, which simplifies the rules as much as possible between successive steps.
3. We employ the training data to simplify the decision list by pruning rules which are not supported by the data.
4. We speed up the computation by caching results of coverage and subsumption of partial rules and reusing these to evaluate combined rules.

We evaluate CoTE on several benchmark relational domains on multiple target relations and show that it produces compact decision lists which are close to the original bagged or boosted decision trees in predictive performance and outperforms an approximation technique that is based on relabeling the training data and learning a single tree.

The rest of the paper is organized as follows: we first review the background and then present the problem of combining logical decision trees. We present the over all algorithm along with two different types of compression methods. Then we evaluate the algorithm empirically before concluding by outlining the areas for future research.

2 Background and Related Work

Compressing ensemble models has a long history in classical machine learning. Typical approaches regenerate data and learn a simpler model (Bastani, Kim, & Bastani, 2017; Vandewiele et al., 2017; Zhou & Hooker, 2016). Other work operates directly on the feature regions (Hara & Hayashi, 2018), while Joly, Schnitzler, Geurts, and Wehenkel (2012) applies Lasso regularization to set of indicator functions to compress ensemble models. While functional gradient boosting Friedman (2001) based approaches have become popular for relational data (Khot, Natarajan, Kersting, & Shavlik, 2015; Natarajan et al., 2012), not much progress has been made on compressing these models to interpretable ones. The key reason is that simple propositional model compression approaches cannot be directly applied to relational domains. Since generating pseudo samples for relational data is difficult, Assche and Blockeel (2007) learns a single first order model to approximate the first order ensemble by exploiting the class distributions predicted by the ensemble.

Some work in propositional domains are closely related to our work. Vidal and Schiffer (2020) uses dynamic programming to find minimal exact tree representation with post pruning, which takes exponential time in the worst case. Our subsumption reduction also finds an exact model. Our example-based pruning is an alternative approach whose output is bounded by the number of training instances although not guaranteed to generate the minimal tree. In addition, our methods preserve numerical scores when combining the trees while other approaches only aim to retain the classification label. Sirikulviriya and Sinthupinyo (2011) performs incremental heuristic rule combination where a rule is a path from the root to a leaf. It defines the conditions of redundancy and conflicts to remove rules that are not necessary to constrain prediction over instance space, which has similarity to our subsumption reduction. Hara and Hayashi (2018) uses independent rules while we use decision lists as explained next.

3 Compression of Logical Decision Trees

Notation: We use uppercase letters for constants and predicate names, and lowercase letters for variables. A (logical) **predicate** is of the form $\mathcal{P}(t_1, \ldots, t_k)$

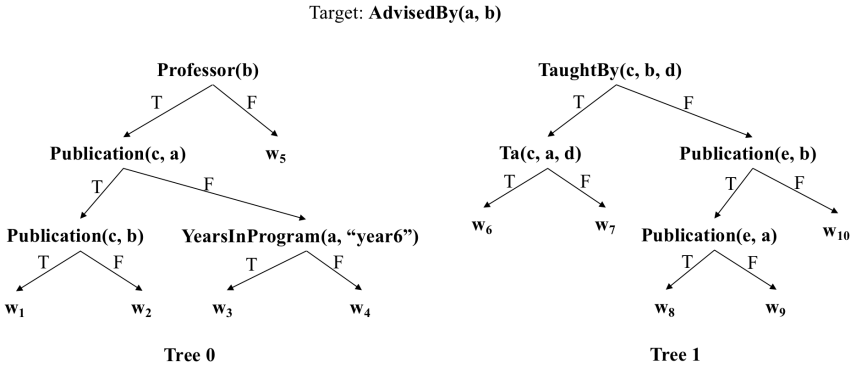

Fig. 1: Example of relational boosted trees. The target relation is **AdvisedBy(a, b)**. Each node contains a relation/predicate as condition. To evaluate a tree, if an instance satisfies the relation in the context of the path from the root, then it move to the left branch, otherwise the right branch.

where $\mathcal{P}$ is a predicate and t_i are **arguments** or logical variables. A **grounding** of a predicate with variables $v_1, \ldots, v_k$ is a substitution $\{\langle v_1, \ldots, v_k \rangle / \langle V_1, \ldots, V_k \rangle\}$ mapping each of its variables to a constant in the domain of that variable.

3.1 Problem Setup and Preliminaries

Given a training set, we use gradient boosting or bagging to generate a set of relational trees (Blockeel & De Raedt, 1998). These TILDE trees include logical variables which are implicitly quantified.

Example 1 As a running example, consider the two trees presented in Figure 1. The goal is to predict if a person **a** is **AdvisedBy b**. The predicates used are **Professor(b)**: **b** is a professor; **Publication(c, a)**: **c** is one of **a**'s publication; **YearsInProgram(a, "year6")**: **a** has been in a program for 6 years; **TaughtBy(c, b, d)**: a course **c** is taught by **b** in a university quarter **d**; **Ta(c, a, d)**: **a** is a teaching assistant of a course **c** in a university quarter **d**. For simplicity of exposition, we will ignore the weights since in the case of boosting, they are combined using the *sum* operation while for bagging, they are combined using *average*.

The semantics of TILDE trees are such that at a node one evaluates the set of positive atoms on the path leading to this node as an existential formula. If this is satisfied then we take the true (left) branch and otherwise the false (right) branch. As a result the formula that describes when a leaf in the tree is reached includes the positive conjunction of all atoms exited via their left branch and the conjunction of negations of existential formulas, one for each false branch taken on the path to the leaf. For example the path formula for leaf w_2 of tree 0 in Figure 1 is $[\exists c\ Professor(b) \wedge Publication(c, a)] \wedge$

$[\neg\exists c\ Professor(b) \wedge Publication(c,a) \wedge Publication(c,b)]$ and the path formula for leaf w_9 of tree 1 is $[\exists e\ Publication(e,b)] \wedge [\neg\exists c,d\ TaughtBy(c,b,d)] \wedge [\neg\exists e\ Publication(e,b) \wedge Publication(e,a)]$. The negated portions imply that the path formulas are mutually exclusive, but they have a complicated form. As we argue below, a relational decision list representation would naturally avoid the complexity of negative predicates in the path formulas, since each rule is applied only when the previous rules are inapplicable. We can now define our task:

Given: A set of m learned logical decision trees $\langle \mathcal{T}_0 ... \mathcal{T}_{m-1} \rangle$.
Task: Return a *compressed* representation that is easily explainable and that *captures* the ensemble model.

Our compressed representation is a decision list $\mathcal{D}$ where each rule in the list is a Horn clause i.e., it has only positive atoms in its body. Such representations have a long history in AI and cognitive science (Klahr, Langley, & Neches, 1986; Laird, 2012; Newell, 1990; Quinlan, 1987). We consider two alternatives for *captures*: in the first, the two representations must be logically equivalent. In the second, we only require the representations to have the same predictions on the original training set that produced the ensemble.

Before presenting our algorithm consider a naive approach to the combination of propositional trees. In this case one can take a cross product of all paths in all trees to generate either one large tree (by hanging tree 1 at each leaf of tree 0, etc for the remaining trees), or a large set of rules that represent these mutually exclusive paths. This approach has two major problems. The first is that the resulting size is exponentially large. This cannot be avoided in the worst case, and in fact it is easy to show that the combination problem is co-NP hard (see Vidal and Schiffer (2020)). The second problem, for the relational case, is that due to first order quantification testing example coverage and testing of implication between rules is computationally hard.

Our algorithm, Compression of (relational) Tree Ensembles (CoTE), mitigates these difficulties through three high-level steps: (1) conversion of the trees to a set of decision lists with only positive atoms, (2) preprocessing the decision lists and caching subsumption and coverage results to speed up the computation, and (3) incremental combination of the trees. In addition, by using example based pruning we guarantee that the output size is always bounded, leading to significant compression and additional speedup.

3.2 Conversion to Positive Decision Lists

The first step is based on the following proposition:

Proposition 1 *A decision list with paths from a TILDE tree given in lexicographical ordering (where True branches are explored before False branches) and where path formulas drop all negative portions is logically equivalent to the original tree.*

Proof First translate the decision tree $\mathcal{T}$ into a decision list $\mathcal{D}$ with negations, where rules are ordered by their lexicographical (left child before right child) ordering. Since paths are mutually exclusive the result is equivalent to the tree. As stated in Section 3.1, every path in the $\mathcal{T}$ is equivalent to a formula of the form $[\exists pos(v_1)] \wedge [\wedge_i \neg \exists neg(v_1, v_{2i})]$ where $pos()$ and $neg()$ are conjunctions of atoms. Our positive $\mathcal{D}$ drops the negated portions from the rules.

Consider case where $\mathcal{D}$ rule is active, i.e., it is satisfied and no rule above it is satisfied. We claim that the corresponding path in the tree is satisfied. First note that the positive portions are identical. Now, by way of contradiction, suppose that one of the negative portions $\exists neg(v_1, v_{2i})$ is true. In that case one of the earlier branches in the tree, whose positive portion is exactly $neg(v_1, v_{2i})$ is satisfied. This implies that one of the earlier rules in the positive $\mathcal{D}$ is satisfied.

Next consider case where a tree path is active. In that case, the positive portion of the corresponding $\mathcal{D}$ rule is satisfied, so we only need to show that no rules above it is active. To see this, note that each positive $\mathcal{D}$ rule above this rule includes a superset of some $\exists neg(v_1, v_{2i})$ from this rule. □

Example 2 Returning to the example, the decision list $\mathcal{D}_0$ corresponding to the first tree $\mathcal{T}_0$ is as follows:

$$w_1 : AdvisedBy(a, b) :- Professor(b) \wedge Publication(c, a) \wedge Publication(c, b)$$
$$w_2 : AdvisedBy(a, b) :- Professor(b) \wedge Publication(c, a)$$
$$w_3 : AdvisedBy(a, b) :- Professor(b) \wedge YearsInProgram(a, \text{“}\texttt{year6}\text{”})$$
$$w_4 : AdvisedBy(a, b) :- Professor(b)$$
$$w_5 : AdvisedBy(a, b) :-$$

Similarly, the second tree $\mathcal{T}_1$ will have five clauses in the decision list $\mathcal{D}_1$ corresponding to the five paths. This produces identical values to the original tree on all possible examples.

As a result, our algorithm uses simple subsumption between sets of positive atoms (De Raedt, Idestam-Almquist, & Sablon, 1997; Plotkin, 1970; Stickel, 1992) to identify rule coverage and implications between different rules. Once the decision list $\mathcal{D}_i$ is created for each tree $\mathcal{T}_i$, we can try to compress the individual clauses using a greedy algorithm that removes one atom at a time and checks for self-subsumption. We note that in our experiments, such a reduction is generally not necessary due to the fact that the learned trees are typically short and do not have many redundancies. After the preprocessing, subsumption is also used to compress combined decision lists.

3.3 Caching Results for Speedup

As noted above, subsumption is NP-complete. Hence we need to ensure that the clauses are as short as possible and it is wise to avoid repeated subsumption/coverage tests. Our caching ideas are based on the following proposition:

Proposition 2 *If a rule has groups of predicates that do not share variables except for variables from the head (target predicate) then the evaluation of the groups can be done separately and combined.*

Proof Fix arguments at the head of the rule to be constants and divide the predicates in the body (clause) into maximal connected subclauses (cf. connected subgraphs), where two predicates are connected if they have a variable (not constant) in common. Then due to disjointness the body is satisfied if and only if all subclauses are satisfied. □

We call these *predicate groups* and use this proposition in two ways. First, whenever possible each clause is divided into predicate groups. Second, a combined rule from two decision lists that are standardized apart is naturally separated into groups. With this in mind, we preprocess the decision lists to generate the following data structures that cache subsumption/example coverage results:

1. **PG** includes all predicate (sub)groups that appear in all clauses in all decision lists. (Algorithm 4)
2. **Subsumption:** A matrix SM is created to store the subsumption results between every pair of predicate groups in PG. (Algorithm 2)
3. **Example Coverage:** The set of examples covered by each predicate group in PG and each clause is stored in ES. (Algorithm 3)

The propositions above imply that, for SCoTE in Section 3.5, subsumption between rules (in this case, combined rules) can be directly resolved through the relationship of their subparts (predicate groups) which is stored in SM.

Caching allows us to actually compute subsumption relationship between predicate groups or training examples covered by each predicate group *only once*. Both operations are very time-consuming. The cached results are used for checking redundancy during the incremental compression.

3.4 Incremental Merging of Lists

The naive algorithm discussed above is indeed naive. Instead of taking a product of all trees in advance and then compressing the results one can perform the operation incrementally. That is we start with tree 0, combine it with tree 1 and compress the results, then continue with tree 2 etc. Therefore, all we need is a correct and effective procedure to combine two decision lists.

One question that arises when combining decision lists is the ordering of the combined rules. This is in contrast with combining sets of mutually exclusive rules where the ordering does not matter. To resolve this we observe the following:

Proposition 3 *The final ordering must preserve the partial order from the original lists but the ordering of "noncomparable" pairs of rules is not important.*

Proof Consider two positive DLs which are standardized apart, ie. their logical variables are disjoint and consider two pairs of rules $\mathcal{A}1 > \mathcal{A}2$ ($\mathcal{A}1$ above $\mathcal{A}2$) and $\mathcal{B}1 > \mathcal{B}2$ on these lists. If an example satisfies both $\mathcal{A}1 \wedge \mathcal{B}2$ and $\mathcal{A}2 \wedge \mathcal{B}1$, then due to disjointness and Proposition 2 the same example also satisfies $\mathcal{A}1 \wedge \mathcal{B}1$. Therefore this example will be active for $\mathcal{A}1 \wedge \mathcal{B}1$ or some rule higher in the combined list and the order of $\mathcal{A}1 \wedge \mathcal{B}2$ and $\mathcal{A}2 \wedge \mathcal{B}1$ will not affect the prediction on this example. □

For example from lists $[\mathcal{A}1, \mathcal{A}2]$ and $[\mathcal{B}1, \mathcal{B}2]$ the combination $(\mathcal{A}1 + \mathcal{B}1)$ must be first and $(\mathcal{A}2 + \mathcal{B}2)$ must be last but the ordering between $(\mathcal{A}1 + \mathcal{B}2)$ and $(\mathcal{A}2 + \mathcal{B}1)$ is not important. The result will be logically equivalent regardless of this ordering. We can therefore explore this cross product in a lexicographical manner. As the following example shows, even a combination of two decision lists can yield a large results. The next two subsections show the result can be compressed during construction.

Example 3 Returning to the example, the combined decision list $\mathcal{D}_{01}$ which contains 25 combinations, corresponding to the cross-product of $\mathcal{D}_0$ and $\mathcal{D}_1$ is as follows for boosting (mean of values for bagging):

$$
\begin{aligned}
w_1 + w_6 &: AdvisedBy(a,b) :- Professor(b) \wedge Publication(c,a) \wedge Publication(c,b) \\
&\quad \wedge TaughtBy(f,b,d) \wedge Ta(f,a,d) \\
w_1 + w_7 &: AdvisedBy(a,b) :- Professor(b) \wedge Publication(c,a) \wedge Publication(c,b) \\
&\quad \wedge TaughtBy(f,b,d) \\
w_1 + w_8 &: AdvisedBy(a,b) :- Professor(b) \wedge Publication(c,a) \wedge Publication(c,b) \\
&\quad \wedge Publication(e,b) \wedge Publication(e,a) \\
w_1 + w_9 &: AdvisedBy(a,b) :- Professor(b) \wedge Publication(c,a) \wedge Publication(c,b) \\
&\quad \wedge Publication(e,b) \\
w_1 + w_{10} &: AdvisedBy(a,b) :- Professor(b) \wedge Publication(c,a) \wedge Publication(c,b) \\
w_2 + w_6 &: AdvisedBy(a,b) :- Professor(b) \wedge Publication(c,a) \\
&\quad \wedge TaughtBy(f,b,d) \wedge Ta(f,a,d) \\
w_2 + w_7 &: AdvisedBy(a,b) :- Professor(b) \wedge Publication(c,a) \wedge TaughtBy(f,b,d) \\
w_2 + w_8 &: AdvisedBy(a,b) :- Professor(b) \wedge Publication(c,a) \wedge Publication(e,b) \\
&\quad \wedge Publication(e,a) \\
w_2 + w_9 &: AdvisedBy(a,b) :- Professor(b) \wedge Publication(c,a) \wedge Publication(e,b) \\
w_2 + w_{10} &: AdvisedBy(a,b) :- Professor(b) \wedge Publication(c,a) \\
w_3 + w_6 &: AdvisedBy(a,b) :- Professor(b) \wedge YearsInProgram(a, \text{“}\texttt{year6}\text{”}) \\
&\quad \wedge TaughtBy(f,b,d) \wedge Ta(f,a,d) \\
w_3 + w_7 &: AdvisedBy(a,b) :- Professor(b) \wedge YearsInProgram(a, \text{“}\texttt{year6}\text{”}) \\
&\quad \wedge TaughtBy(f,b,d) \\
w_3 + w_8 &: AdvisedBy(a,b) :- Professor(b) \wedge YearsInProgram(a, \text{“}\texttt{year6}\text{”}) \\
&\quad \wedge Publication(e,b) \wedge Publication(e,a) \\
w_3 + w_9 &: AdvisedBy(a,b) :- Professor(b) \wedge YearsInProgram(a, \text{“}\texttt{year6}\text{”}) \\
&\quad \wedge Publication(e,b)
\end{aligned}
$$

$$
\begin{aligned}
w_3 + w_{10} &: AdvisedBy(a,b) :- Professor(b) \wedge YearsInProgram(a, \text{“}\texttt{year6}\text{”}) \\
w_4 + w_6 &: AdvisedBy(a,b) :- Professor(b) \wedge TaughtBy(f,b,d) \wedge Ta(f,a,d) \\
w_4 + w_7 &: AdvisedBy(a,b) :- Professor(b) \wedge TaughtBy(f,b,d) \\
w_4 + w_8 &: AdvisedBy(a,b) :- Professor(b) \wedge Publication(e,b) \wedge Publication(e,a) \\
w_4 + w_9 &: AdvisedBy(a,b) :- Professor(b) \wedge Publication(e,b) \\
w_4 + w_{10} &: AdvisedBy(a,b) :- Professor(b) \\
w_5 + w_6 &: AdvisedBy(a,b) :- TaughtBy(f,b,d) \wedge Ta(f,a,d) \\
w_5 + w_7 &: AdvisedBy(a,b) :- TaughtBy(f,b,d) \\
w_5 + w_8 &: AdvisedBy(a,b) :- Publication(e,b) \wedge Publication(e,a) \\
w_5 + w_9 &: AdvisedBy(a,b) :- Publication(e,b) \\
w_5 + w_{10} &: AdvisedBy(a,b) :-
\end{aligned}
$$

3.5 Subsumption based Compression

For the algorithm in this section, our goal is to compress the decision list but maintain logical equivalence. This relies on subsumption tests. In particular, compression is used in two places. The first compression is within a rule where we can remove individual predicates (during preprocessing) or subgroups of predicates during the combination of rules. For instance, in the above example, the $(w_1 + w_8)$ clause can drop $Publication(e,a)$ and $Publication(e,b)$ and be reduced to (similar to the $(w_1 + w_9)$ clause)

$$
\begin{aligned}
w_1 + w_8 : AdvisedBy(a,b) :- & Professor(b) \wedge Publication(c,a) \\
& \wedge Publication(c,b). \qquad (1)
\end{aligned}
$$

The second compression is removal of rules which are subsumed by rules that appear above them in the list. As mentioned above all these decisions can be done directly using SM without rerunning subsumption tests. For instance, the clause in Equation 1 subsumes the $(w_1 + w_{10})$ clause (as well as the $(w_1 + w_9)$ clause) from Example 3. So now the subsumption reduction is applied and the decision list will be simplified to (we show only rules with w_1 for brevity)

$$
\begin{aligned}
w_1 + w_6 : AdvisedBy(a,b) :- & Professor(b) \wedge Publication(c,a) \\
& \wedge Publication(c,b) \wedge TaughtBy(f,b,d) \\
& \wedge Ta(f,a,d) \\
w_1 + w_7 : AdvisedBy(a,b) :- & Professor(b) \wedge Publication(c,a) \\
& \wedge Publication(c,b) \wedge TaughtBy(f,b,d) \\
w_1 + w_8 : AdvisedBy(a,b) :- & Professor(b) \wedge Publication(c,a) \\
& \wedge Publication(c,b) \\
& \ldots\ldots
\end{aligned}
$$

We refer to this algorithm that uses subsumption for reduction as SCoTE. As shown in our experiments this algorithm produces a logically equivalent expression that provides a significant compression relative to the naive algorithm.

3.6 Example based Compression

While subsumption often produces small lists this is not always the case, and we know that in the worst case with arbitrary trees the results can be of exponential size. However, recall that we are generating a decision list from the original given dataset. If the dataset has N examples, then the final representation should not have more than N distinctions, N rules in cases of a list and N leaves if we were using a tree. If the intermediate representation has more rules, then the cases they cover have not been validated by the data. They either correspond to situations that do not naturally arise in the data, or their prediction has not been validated and is hence not trustworthy. Although we are preforming a logical transformation of the ensemble, our idea is to: *prune rules in the decision list which do not cover any example in the original dataset.* If the original ensemble has good predictions they are likely from the retained rules so we expect to retain predictive accuracy while avoiding excessive size.

The incremental algorithm works in exactly the same manner as in SCoTE. But the decisions on compression are based on the coverage of examples instead of subsumption between rules. To distinguish from SCoTE, we refer to this algorithm as ECoTE. While SCoTE preserves logical equivalence with the learned ensemble, ECoTE guarantees that training examples coverage is the same as the original model.

ECoTE can reduce rules further than SCoTE. For example, in our training data, any person who is a professor has some publication. Then removing $Publication(e, b)$ does not affect example coverage of the $(w_2 + w_9)$ clause. Hence we get

$$w_2 + w_9 : AdvisedBy(a, b) :- Professor(b) \wedge Publication(c, a)$$

ECoTE also detects combined rules with empty coverage relative to the dataset. Such rules are needed to maintain logical equivalence but not relative to our dataset. For example, professors publish papers with students in departments and teach courses. The students publish papers with a professor always have been a teaching assistant in at least one of the professor's course. Under this situation, the $(w_1 + w_6)$ clause covers all positive examples that the $(w_1 + w_7)$ clause can cover. If an example fails the $(w_1 + w_6)$ clause, it will also fails the $(w_1 + w_7)$ clause. Hence the example coverage of the clause $(w_1 + w_7)$ is empty and the clause can be removed.

3.7 Algorithm

Combining all the observations above we get our promised compression algorithms. Algorithm 1 presents the *Compression of Tree Ensembles* (CoTE)

Algorithm 1 CoTE: **Co**mpression of **T**ree **E**nsembles

```
1: procedure CoTE(Ensemble model M)
2:     L, PG ← PREP(M)                                   ▷ See Alg 4
3:     if logic-equivalent then
4:         L ← SR(L, PG)                                 ▷ See Alg 2
5:     else if data-equivalent then
6:         L ← EP(L, PG)                                 ▷ See Alg 3
7:     end if
8:     return L
9: end procedure
```

Algorithm 2 SR: **S**ubsumption **R**eduction

```
1: procedure SR(Decision lists L, Predicate groups PG)
2:     M ← size(PG)
3:     for i ∈ 0 to M − 1 do
4:         for j ∈ 0 to M − 1 do
5:             SM_{i,j} ← isSubsume(PG_i, PG_j)
6:         end for
7:     end for
8:     N ← size(L)
9:     L ← L_0
10:    for iter i ∈ 1 to N − 1 do
11:        L* ← ∅
12:        for each C_j ∈ L do
13:            for each C_k ∈ L_i do
14:                C ← addClauses(C_j, C_k)
15:                L* ← append(L*, reduceClause(C, SM))   ▷ In clause
16:            end for
17:        end for
18:        L ← reduceList(L*, SM)                         ▷ Between clauses
19:    end for
20:    return L
21: end procedure
```

algorithm. Preprocessing into decision lists and predicates groups is performed in Algorithm 4. Algorithm 2 describes SCoTE. Lines 3-7 build a matrix to store the subsumption relationship between any pair of predicate groups. In line 15, the combined clause is shortened by checking subsumption relationship between its predicate groups before it is appended to our decision list. In line 18, clauses that are subsumed by any clause higher in the decision list are removed. Algorithm 3 describes ECoTE. In line 5, the algorithm generates example coverage of not only each clause in each decision list, but also each predicate group in each clause. The example coverage of each clause is later used in line 14 to remove combined clauses with empty intersection of example coverage while also updates the overall example coverage EC and record the

Algorithm 3 EP: **E**xample based **P**runing

```
1: procedure EP(Decision lists L, Predicate groups PG)
2:     ES ← ∅                                              ▷ Examples coverage of PG
3:     N ← size(L)
4:     for i ∈ 0 to N − 1 do
5:         ES ← append(ES, expCover(D_train, L_i, PG))
6:     end for
7:     L ← L_0
8:     for iter i ∈ 1 to N − 1 do
9:         L* ← ∅
10:        EC ← ∅                                          ▷ Examples coverage
11:        for each C_j ∈ L do
12:            for each C_k ∈ L_i do
13:                C ← addClauses(C_j, C_k)
14:                EC_C ← checkCoverage(C, ES, EC)         ▷ Between clauses
15:                if EC_C ≠ ∅ then
16:                    L* ← append(L*, pruneClause(C, ES, EC_C, EC))   ▷ In clause
17:                end if
18:            end for
19:        end for
20:        L ← L*
21:    end for
22:    return L
23: end procedure
```

example coverage of the clause $\mathtt{C}$ as $EC_{\mathtt{C}}$. Similarly, example coverage of predicate groups helps check whether removing a predicate group will change the example coverage of the combined clause $\mathtt{C}$ in line 13. ES, $EC_{\mathtt{C}}$ and EC are used in line 16 to prune the clause $\mathtt{C}$ without changing the example coverage of $\mathtt{C}$ before appending this clause to L^*. Algorithm 4 shows the preprocessing procedure which is common for both subsumption reduction and example based pruning. This procedure transforms the trees to a decision list, iterates through each rule, reduces it and constructs the groups of literals.

4 Experimental Evaluation

We aim to answer the following questions explicitly: **Q1:** Are our compression methods effective in improving explainability by reducing the number of clauses and the average clause length? **Q2:** Are our compression methods faithful to the original model? **Q3:** How do the methods compare against re-labeling the data and fitting a (logical) decision tree? To answer these questions, we consider 5 standard relational data sets. We compare our SCoTE and ECoTE algorithms against a re-labeling method where the training data is re-labeled by the learned ensemble and a single tree is induced from this re-labeled data. We call this as SOFT LABELS Craven and Shavlik (1995). This is a strong baseline and is the one most often used for relational learning (Khot et al., 2015; Natarajan et al., 2012).

Algorithm 4 Preprocess the ensemble model

procedure PREP(Ensemble model $\mathcal{M}$)
  $N \leftarrow$ size($\mathcal{M}$)
  $\mathcal{L} \leftarrow \emptyset$
  $PG \leftarrow \emptyset$
  for $i \in 0$ **to** $N - 1$ **do**
    $L \leftarrow$ transform($\mathcal{M}_i$) ▷ Tree to list
    $L^* \leftarrow \emptyset$
    for each $\mathtt{C}_j$ in L **do**
      $\mathtt{C}_j \leftarrow$ clauseReduction($\mathtt{C}_j$) ▷ Reduce clause
      $L^* \leftarrow$ append(L^*, $\mathtt{C}_j$)
      $PG \leftarrow$ append(PG, literalGroups($\mathtt{C}_j$))
    end for
    $\mathcal{L} \leftarrow$ append($\mathcal{L}$, L^*)
  end for
  return $\mathcal{L}$, PG
end procedure

Datasets: (1) The **UW-CSE** data set Richardson and Domingos (2006) is a standard benchmark where the task is to predict whether a professor and student share an advisor-advisee relationship based on other relationships involving professors, students, courses, publications, classes etc., in 5 areas of Computer Science. (2) **Cora Entity Resolution** is a dataset describing the relationship between citations. The task is to identify which citations refer to the same publication. (3) The **IMDB** data set was first created by Mihalkova and Mooney (2007) and contains relationships among movie elements like cast, genre etc. (4) The **WebKB** data set consists of web pages and hyperlinks from 4 CS departments (Slattery & Craven, 1998) and the task is to predict if someone is a faculty member. (5) The **ICML** data set, extracted from the Microsoft academic graph (MAG) (Dhami, Yan, Kunapuli, & Natarajan, 2021; Sinha et al., 2015), consists of papers from ICML 2018 and the task is to predict coauthors.

Experiment setup: We ran Boosting and Bagging algorithms with the number of trees set as 20 to yield two sets of decision trees. We then ran compression methods SCoTE and ECoTE on both Boosting and Bagging trees. We also ran soft labels method on Boosting trees. For Boosting, the size of each tree is small as weak learner. For Bagging, large TILDE trees are needed for better performance.

Results: Table 1 presents the results of compressing the boosted model with 20 TILDE trees in all the domains and compares SCoTE and ECoTE with soft labels. As can be noted, a simple combination can yield an exponential number of rules with large rule lengths. In all but two of the domains, SCoTE was able to achieve significant compression. In two domains, the runtime for SCoTE is excessive and it did not complete the compression in

Table 1: Number of clauses/rules in relational domains and average rule length when the base model is **Boosting** with 20 TILDE trees. Note that while subsumption compresses the model significantly, the gains are higher when using the example based reduction. "Max clauses" and "avg" length of original model are the results when the clauses are naively combined without any compression; "max clauses" is the maximal possible number of clauses of combining 20 trees which is also the worst case when no compression can be applied; "avg" is the average clause length in this case; "#paths" is the total number of paths of the original 20 trees; "depth" is the maximal depth over the 20 trees. AUC scores close or higher to original model are bolded.

Domain	Relation	Original model						SCoTE				ECoTE				Soft labels			
		#paths	depth	max clauses	avg	ROC	PR	#clause	avg	ROC	PR	#clause	avg	ROC	PR	#clause	avg	ROC	PR
Cora	Same-Author(A,B)	99	5	7.6×10^{13}	58.10	0.726	0.941	18	6.83	**0.726**	**0.941**	9	4.33	**0.726**	**0.941**	5	3.00	0.673	0.938
	Same-Bib(A,B)	80	4	1.1×10^{12}	36.50	0.921	0.949	7	2.86	**0.921**	**0.949**	7	2.86	**0.921**	**0.949**	4	2.00	0.918	**0.948**
	Same-Title(A,B)	80	6	1.1×10^{12}	38.75	0.693	0.672	30	6.33	**0.693**	**0.672**	14	3.86	**0.693**	**0.672**	4	1.75	0.684	0.656
	Same-Venue(A,B)	100	7	9.5×10^{13}	56.40	0.806	0.694	238	10.70	**0.806**	**0.694**	52	6.23	**0.805**	0.690	5	3.20	0.553	0.416
WebKB	Faculty(A)	129	7	1.2×10^{16}	42.58	0.678	0.474	96	10.54	**0.678**	**0.474**	29	4.93	**0.678**	**0.474**	8	2.38	0.592	0.231
	Course-Prof(A,B)	175	6	6.3×10^{18}	49.60	0.521	0.365	36	4.19	**0.521**	**0.365**	18	2.78	**0.521**	**0.365**	9	2.22	0.491	0.333
	Course-TA(A,B)	164	7	1.5×10^{18}	56.60	0.575	0.439	31	5.35	**0.575**	**0.439**	12	2.83	0.548	0.374	7	3.71	0.540	0.384
UW-CSE	Advised-By(A,B)	157	4	5.5×10^{17}	35.52	0.997	0.994	n/a	n/a	n/a	n/a	103	2.95	**0.996**	**0.993**	9	2.22	0.943	0.927
IMDB	Worked-Under(A,B)	81	3	1.4×10^{12}	30.10	1.000	1.000	6	2.00	**1.000**	**1.000**	5	1.60	**1.000**	**1.000**	4	1.50	**1.000**	**1.000**
	Genre(A,B)	100	1	9.5×10^{13}	16.00	0.620	0.452	512	4.50	**0.620**	**0.452**	9	0.89	**0.620**	**0.452**	5	0.80	0.419	0.178
	Female-Gender(A)	133	4	2.5×10^{16}	26.70	0.655	0.633	1591	5.32	0.655	**0.633**	13	1.00	**0.657**	**0.634**	6	1.67	0.626	0.542
ICML	Co-Author(A,B)	259	4	1.8×10^{22}	31.03	0.844	0.275	n/a	n/a	n/a	n/a	3972	3.18	**0.801**	**0.269**	13	1.46	0.501	0.022

10 hours. We also note from the results that ECoTE is significantly better than SCoTE in a majority of the domains. Similar results can be observed when compressing bagged models with 20 large TILDE trees in Table 2 where SCoTE achieves significant compression in a majority of domains while ECoTE is even better.

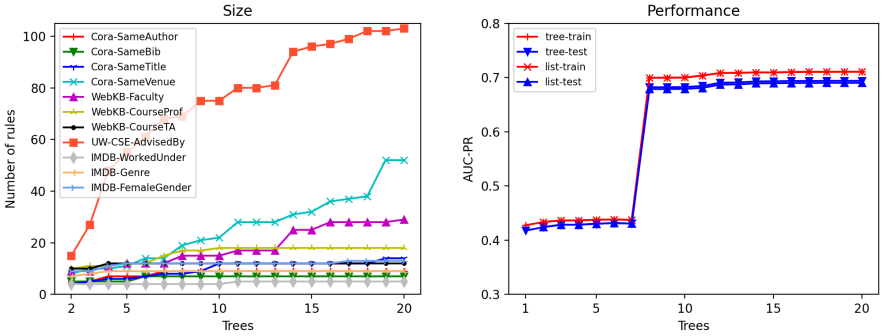

Fig. 2: Sample results for example based compression on gradient boosted trees. (a) Number of rules vs. number of trees for 11 relational domains. (b) Cora SameVenue task. Note that the list's train and test set performance are not significantly worse than the tree performance

A sample of these results can be seen in Figure 2. Figure 2.*a* presents the results of plotting the number of rules as a function of the number of trees for ECoTE. Note that we do not plot ICML because the number of rules are quite high and it will render all the other curves flat. This shows that in most problem we are able to compress into a small number of rules regardless of the number of trees combined. The performance in a single relation of CORA is presented in Figure 2.*b*. This illustrates that Boosting is needed (because at least 8 trees are needed to obtain good performance), that there is no overfitting because train and test performance are close to each other, and that the decision list matches the performance of the ensemble of trees. More performance curves are presented in the appendix. In summary, **Q1** can be answered affirmatively in that the algorithms achieve significant compression over simple combinations.

To understand the effectiveness, we present the test set performance of the compression algorithms in Table 1 for boosted models and in Table 2. Specifically, we present the AUCPR and AUCROC obtained by the original models (boosting with 20 trees and bagging with 20 trees respectively) and compare their performance with SCoTE and ECoTE. It can be observed that as expected SCoTE preserved the predictions of the ensemble exactly, and the score for ECoTE are not significantly different, thus answering **Q2** affirmatively.

Table 2: Number of clauses/rules in relational domains and average rule length when the base model is **Bagging** with 20 large TILDE trees. The results are similar to the boosted combination where ECoTE is more efficient than SCoTE without significance loss in performance. Column names refer to Table 1. AUC scores close or higher to original model are bolded.

Domain	Relation	Original model						SCoTE				ECoTE			
		#paths	depth	Max clauses	avg	ROC	PR	#clause	avg	ROC	PR	#clause	avg	ROC	PR
Cora	Same-Author(A,B)	66	7	2.3×10^{7}	29.14	0.676	0.944	51	7.24	**0.676**	**0.944**	23	4.96	**0.676**	**0.944**
	Same-Bib(A,B)	164	11	1.1×10^{16}	60.25	0.927	0.960	141	12.86	**0.927**	**0.960**	40	6.70	**0.926**	**0.959**
	Same-Title(A,B)	47	5	9.7×10^{4}	17.41	0.656	0.640	48	7.21	**0.656**	**0.640**	15	3.93	**0.656**	**0.640**
	Same-Venue(A,B)	95	9	1.2×10^{9}	35.29	0.573	0.433	492	13.66	**0.573**	**0.433**	65	6.22	**0.572**	0.431
WebKB	Faculty(A)	81	7	2.8×10^{10}	23.01	0.696	0.514	52	6.52	**0.696**	**0.514**	29	5.00	**0.696**	**0.514**
	Course-Prof(A,B)	100	7	7.5×10^{11}	31.68	0.506	0.361	45	5.11	**0.506**	**0.361**	19	2.89	**0.507**	**0.361**
	Course-TA(A,B)	50	7	1.2×10^{6}	17.29	0.490	0.338	16	3.44	**0.490**	**0.338**	9	2.56	**0.490**	**0.338**
UW-CSE	Advised-By(A,B)	325	7	1.5×10^{24}	48.76	0.987	0.975	n/a	n/a	n/a	n/a	126	2.42	**0.986**	0.956
IMDB	Worked-Under(A,B)	85	8	2.4×10^{10}	31.42	1.000	1.000	540	8.79	**1.000**	**1.000**	42	3.24	**1.000**	**1.000**
	Genre(A,B)	65	5	1.2×10^{8}	14.68	0.709	0.483	6444	8.73	**0.709**	**0.483**	11	1.36	**0.709**	**0.483**
	Female-Gender(A,B)	132	6	2.9×10^{13}	35.88	0.701	0.646	n/a	n/a	n/a	n/a	18	1.72	0.680	0.635
ICML	Co-Author(A,B)	196	4	3.4×10^{16}	25.71	0.705	0.155	n/a	n/a	n/a	n/a	1738	2.99	0.684	0.110

While it appears that the soft label generation method has significantly fewer clauses with smaller lengths, it suffers from two issues – being approximate, its performance is sometimes significantly worse than SCoTE and ECoTE. In nearly all the domains, its AUCPR and AUCROC is worse than the compression methods. The second issue is that the final learned model is not representative (neither logically equivalent nor coverage equivalent) of the original model. Hence, it cannot be directly used for interpretation of the original model. Thus **Q3** can be strongly answered in that the compression techniques significantly outperform the soft label strategy.

The discussion so far provided a quantitative evaluating of the compression algorithms. We next illustrate the compression qualitatively by showing a concrete outcome. The final combined decision list of $SameBib(a, b)$ (denoted as *SB*) using ECoTE to compress 20 boosting trees is presented below (*T - Title, V - Venue, A - Author, HV - HasWordVenue, HA - HasWordAuthor*). As can be seen the rules are compact and interpretable. This clearly demonstrates that it is possible to convert a potential blackbox model such as relational boosted trees and generate an interpretable and explainable model.

$$
\begin{aligned}
&4.462 : SB(a,b) :- T(a,c) \wedge T(b,c).\\
&2.684 : SB(a,b) :- V(b,c) \wedge V(a,c) \wedge A(b,d) \wedge HV(e,f) \wedge \\
&\qquad\qquad\qquad A(a,d) \wedge HA(d,f).\\
&2.667 : SB(a,b) :- V(b,c) \wedge V(a,c) \wedge A(b,d) \wedge A(a,d).\\
&2.489 : SB(a,b) :- V(b,c) \wedge V(a,c).\\
&0.158 : SB(a,b) :- A(b,c) \wedge HV(d,e) \wedge A(a,c) \wedge HA(c,e).\\
&0.141 : SB(a,b) :- A(a,c) \wedge A(b,c).\\
&0.198 : SB(a,b) :- .
\end{aligned}
$$

While successful, both SCoTE and ECoTE can be less efficient in certain conditions. For instance, in *ICML*, the learned model mainly had constants in the tree nodes (as opposed to variables in other data sets). This renders the subsumption powerless as the constants cannot subsume other constants. Consequently, SCoTE did not converge/finish even after 10 hours. While ECoTE does not rely on subsumption and was able to converge/finish, it still generated large number of rules (~ 4000). The investigation of the efficiency of these methods on purely propositional data sets (where the models employ only constants) remains an interesting direction for future research. It is clear that from our model construction and our experiments that *relational problems are more amenable to compression* due to the inherent parameterization.

5 Conclusion

We have presented SCoTE and ECoTE, two compression algorithms based on subsumption and example based reduction to compress relational ensemble models. While SCoTE preserves logical equivalence, ECoTE preserves the

example coverage and thus both methods do not suffer a loss in performance due to compression. We presented the algorithms from a classification perspective but it should be noted that this can be extended to perform regression or even algebraic operations of trees such as product, addition or max operations. This will allow the learnable boosted/bagged models to be used for relational reinforcement learning. Solving relational Bellman operator (REBEL) Joshi, Kersting, and Khardon (2009); Kersting, Otterlo, and De Raedt (2004); Sanner and Boutilier (2009); Wang, Joshi, and Khardon (2008) can allow for building relational RL algorithms efficiently. Understanding the relationships of the relational compression algorithms to classical propositional compression ones is an interesting future research direction.

Supplementary information. Please see Appendix A.

Acknowledgments. Siwen Yan and Sriraam Natarajan acknowledge DARPA Minerva award FA9550-19-1-0391. Sriraam Natarajan also acknowledges the support of ARO award W911NF2010224. Prasad Tadepalli acknowledges support of DARPA contract N66001-17-2-4030 and NSF & USDA-NIFA under grant 2021-67021-35344. Roni Khardon acknowledges support of NSF under grant IIS-2002393.

References

Assche, A.V., & Blockeel, H. (2007). Seeing the forest through the trees: Learning a comprehensible model from an ensemble. *Ecml.*

Bastani, O., Kim, C., Bastani, H. (2017). Interpreting blackbox models via model extraction. *arXiv:1705.08504*.

Blockeel, H., & De Raedt, L. (1998). Top-down induction of first-order logical decision trees. *AI*.

Craven, M., & Shavlik, J. (1995). Extracting tree-structured representations of trained networks. *NeurIPS*.

De Raedt, L., Kersting, K., Natarajan, S., Poole, D. (2016). *Statistical relational artificial intelligence logic, probability, and computation.* San Rafael: Morgan & Claypool.

De Raedt, L., Idestam-Almquist, P., Sablon, G. (1997). θ-subsumption for structural matching. *Ecml.*

Dhami, D.S., Yan, S., Kunapuli, G., Natarajan, S. (2021). Non-parametric learning of embeddings for relational data using gaifman locality theorem. *Ilp.*

Friedman, J.H. (2001). Greedy function approximation: A gradient boosting machine. *Ann. Stat.*.

Hara, S., & Hayashi, K. (2018). Making tree ensembles interpretable: A bayesian model selection approach. *Aistats.*

Hinton, G., Vinyals, O., Dean, J. (2015). *Distilling the knowledge in a neural network.*

Joly, A., Schnitzler, F., Geurts, P., Wehenkel, L. (2012). L1-based compression of random forest models. *Esann.*

Joshi, S.S., Kersting, K., Khardon, R. (2009). Generalized first order decision diagrams for first order markov decision processes. *Twenty-first international joint conference on artificial intelligence.*

Kersting, K., Otterlo, M.V., De Raedt, L. (2004). Bellman goes relational. *Proceedings of the twenty-first international conference on machine learning* (p. 59).

Khot, T., Natarajan, S., Kersting, K., Shavlik, J. (2015). Gradient-based boosting for statistical relational learning: The markov logic network and missing data cases. *MLJ*.

Klahr, D., Langley, P., Neches, R. (1986). *Production system models of learning and development*. Cambridge, MA: MIT press.

Laird, J.E. (2012). *The soar cognitive architecture*. Cambridge: MIT Press.

Mihalkova, L., & Mooney, R.J. (2007). Bottom-up learning of markov logic network structure. *Icml.*

Natarajan, S., Khot, T., Kersting, K., Gutmann, B., Shavlik, J. (2012). Gradient-based boosting for statistical relational learning: The relational dependency network case. *MLJ*.

Natarajan, S., Khot, T., Kersting, K., Shavlik, J. (2015). *Boosted statistical relational learners: From benchmarks to data-driven medicine*. New York: Springer.

Newell, A. (1990). *Unified theories of cognition*. Cambridge: Harvard University Press.

Plotkin, G.D. (1970). A note on inductive generalization. *Machine Intelligence*.

Quinlan, J.R. (1987). Generating production rules from decision trees. *Ijcai.*

Richardson, M., & Domingos, P. (2006). Markov logic networks. *ML.*

Sanner, S., & Boutilier, C. (2009). Practical solution techniques for first-order mdps. *Artificial Intelligence*, *173*(5-6), 748–788.

Sinha, A., Shen, Z., Song, Y., Ma, H., Eide, D., Hsu, B., Wang, K. (2015). An overview of microsoft academic service (mas) and applications. *Www.*

Sirikulviriya, N., & Sinthupinyo, S. (2011). Integration of rules from a random forest. *Iciee.*

Slattery, S., & Craven, M. (1998). Combining statistical and relational methods for learning in hypertext domains. *Ilp.*

Stickel, M.E. (1992). A prolog technology theorem prover: A new exposition and implementation in prolog. *TCS*.

Vandewiele, G., Lannoye, K., Janssens, O., Ongenae, F., De Turck, F., Van Hoecke, S. (2017). A genetic algorithm for interpretable model extraction from decision tree ensembles. *Pakdd.*

Vidal, T., & Schiffer, M. (2020). Born-again tree ensembles. *Icml.*

Wang, C., Joshi, S., Khardon, R. (2008). First order decision diagrams for relational mdps. *Journal of Artificial Intelligence Research*, *31*, 431–472.

Zhou, Y., & Hooker, G. (2016). Interpreting models via single tree approximation. *arXiv:1610.09036*.

Appendix A Performance

Figures A1 through A12 of the appendix present the train and test set performances of the original boosted trees and the learned decision lists. It can be observed that in all the domains the performance between the original trees and the compressed decision lists are not significantly different. In addition, a majority of domains clearly show the advantage of boosting where the performance increases as the number of trees increase, thus justifying the necessity of learning a powerful model and then deriving a compact representation of these models.

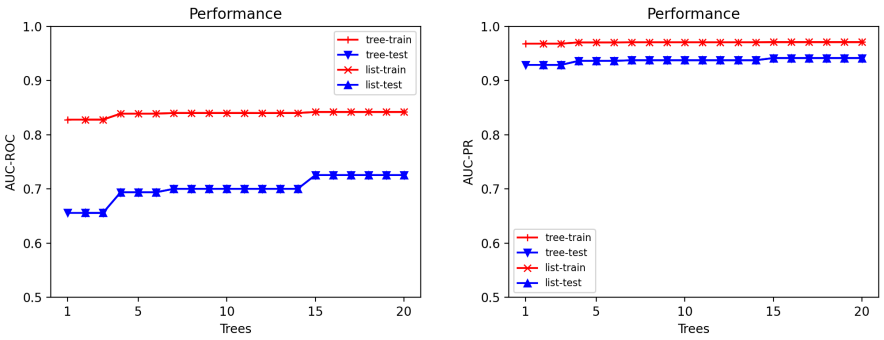

Fig. A1: Cora SameAuthor with boosting

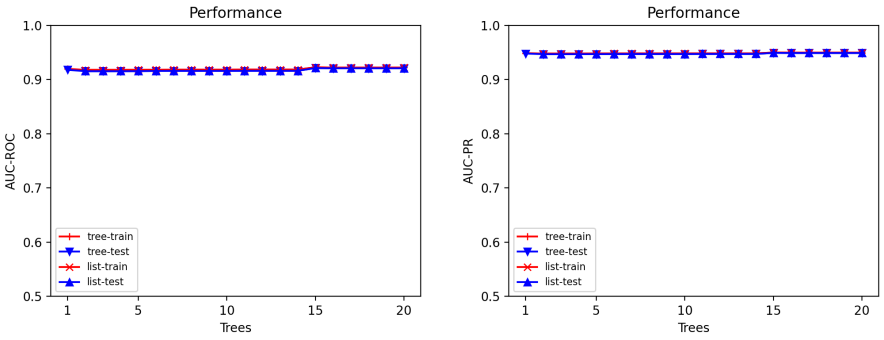

Fig. A2: Cora SameBib with boosting

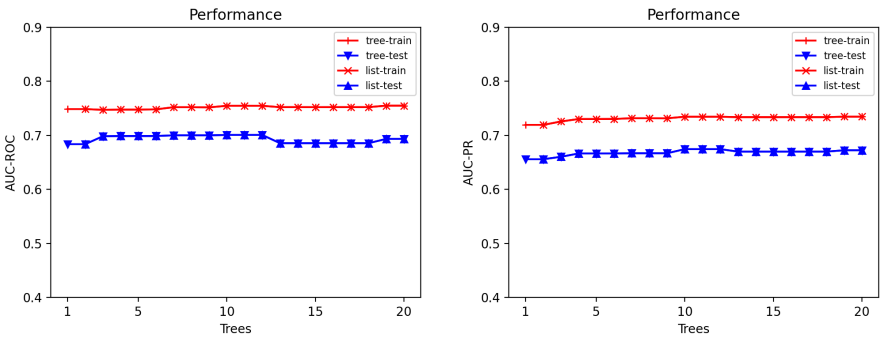

Fig. A3: Cora SameTitle with boosting

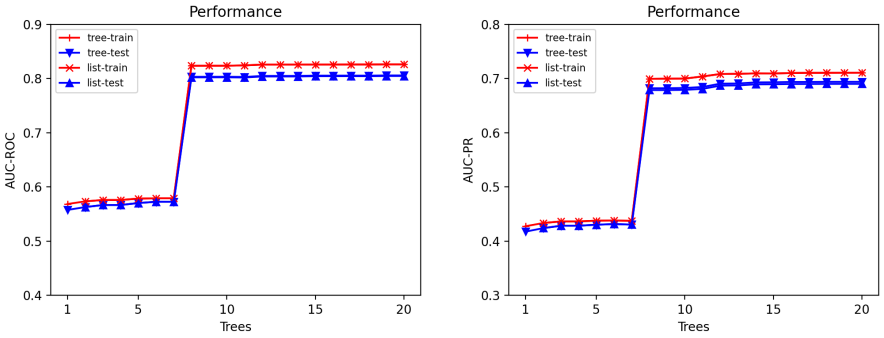

Fig. A4: Cora SameVenue with boosting

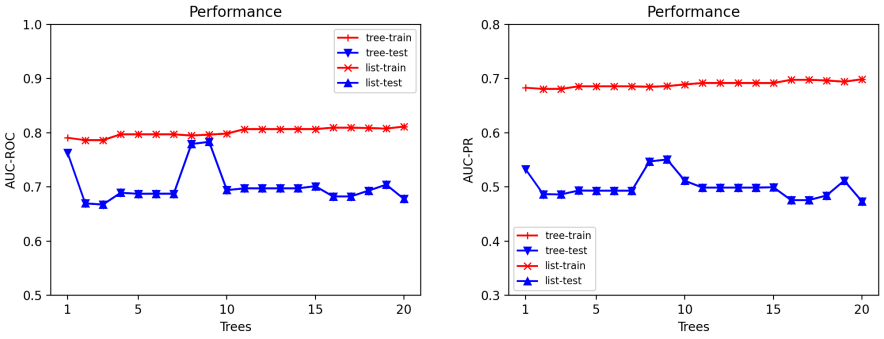

Fig. A5: WebKB Faculty with boosting

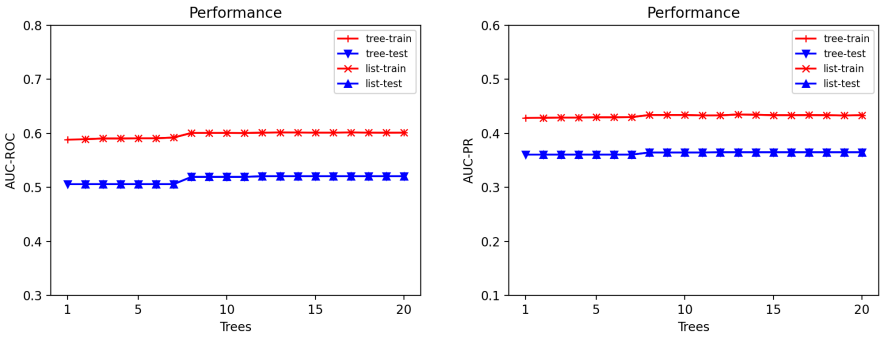

Fig. A6: WebKB CourseProf with boosting

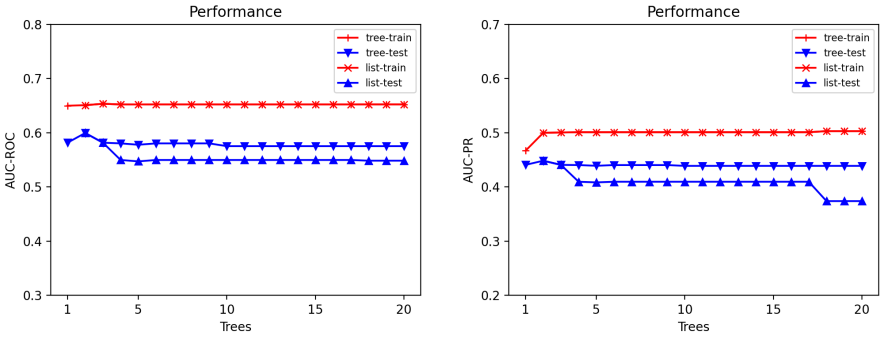

Fig. A7: WebKB CourseTA with boosting

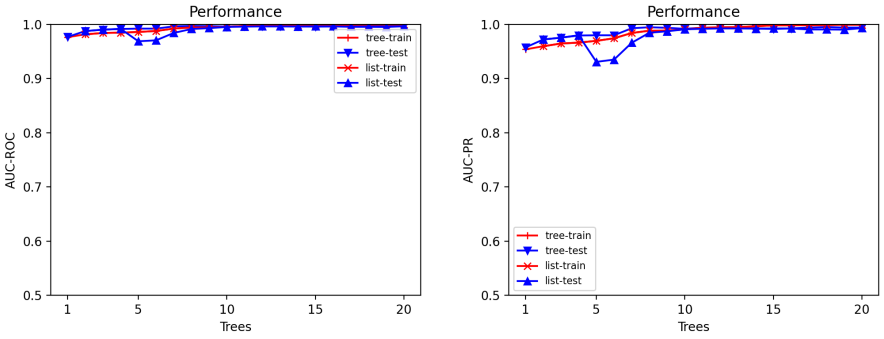

Fig. A8: UW-CSE AdvisedBy with boosting

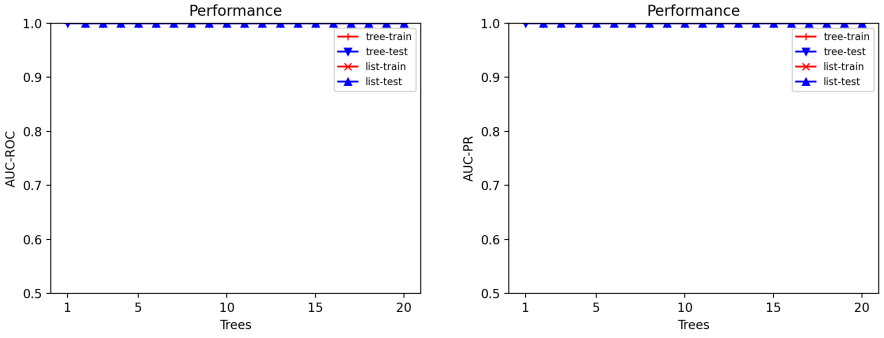

Fig. A9: IMDB WorkedUnder with boosting

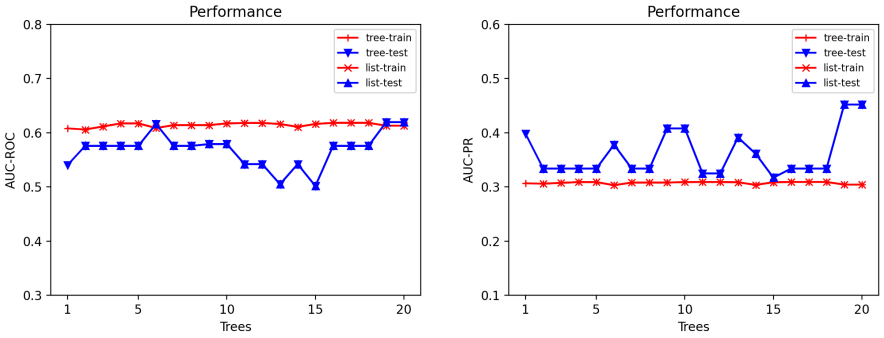

Fig. A10: IMDB Genre with boosting

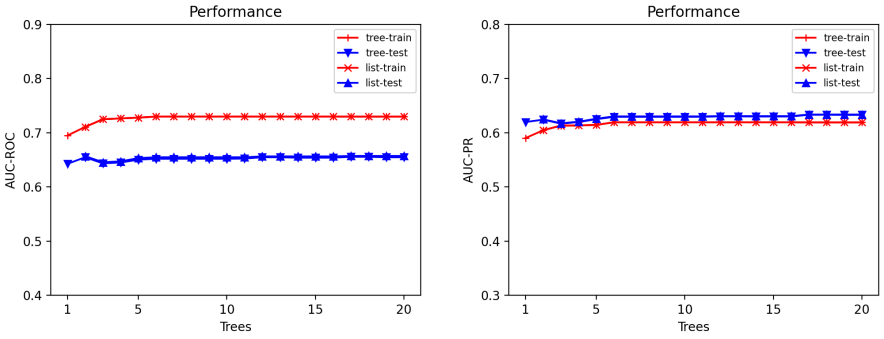

Fig. A11: IMDB FemaleGender with boosting

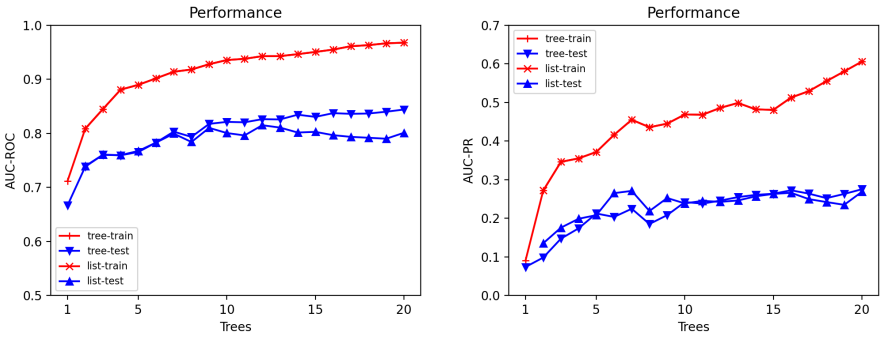

Fig. A12: ICML CoAuthor with boosting