Procedure 1 Structure Learning using relOCC; **Input**: fact base $\mathbf{F_s}$, positive ex. pos, negative ex. neg

function `LearnGaifmanStruct`($\mathbf{F_s}$, pos, neg)
    for every $\mathbf{x_1}, \mathbf{x_2}$ **in** pos **do**
        Calculate $d(x_1, x_2)$
        $D(x_1,\, x_2) = \sum_i \, \beta_i \, d_i(x_1, x_2)$
        ▷ Compute weighted distance between x_1, x_2
    end for
    for a given new unlabeled example $\mathbf{z}$ **do**
        $E(z \notin \texttt{class}) \; = \; \sum_j \; \alpha_j D(x_j,\, z)$
        ▷ Calculate the density estimate
    end for
    Learn the tree `T` iteratively
    return `LeftBranch`(`T`)
    ▷ Obtain the positive density relational rules
    for every $\mathbf{x'_1}, \mathbf{x'_2}$ **in** neg **do**
        Calculate $d(x'_1, x'_2)$
        $D(x'_1,\, x'_2) = \sum_i \, \beta_i \, d_i(x'_1, x'_2)$
        ▷ Compute weighted distance between x'_1, x'_2
    end for
    for a given new unlabeled example $\mathbf{z'}$ **do**
        $E(z' \notin \texttt{class}) \; = \; \sum_j \; \alpha_j D(x'_j,\, z')$
        ▷ Calculate the density estimate
    end for
    Learn the tree `T` iteratively
    return `LeftBranch`(`T`)
    ▷ Obtain the negative density relational rules
end function

Procedure 2 Learning Gaifman Embeddings; **Input**: target query $\mathbf{q}$, knowledge base $\mathcal{B}$, positives pos, negatives neg; **Params**: depth r, size k and number of Gaifman neighborhoods w

1: **function** LGE($\mathbf{q}$, $\mathcal{B}$, pos, neg)
2: $\mathcal{G}$ = `MakeGaifmanGraph`($\mathcal{B}$)
3: ▷ construct Gaifman graph from facts
4: F_s = `MakeFactBase`($\mathcal{B}$)
5: Φ = `LearnGaifmanStruct`(F_s, pos, neg)
6: ▷ extract relational features
7: G_{pos}, G_{neg} = `Ground`(Φ, F_s, pos)
8: ▷ ground positive and negative examples
9: $T_{\mathbf{q}}^{\mathsf{pos}}$, $T_{\mathbf{q}}^{\mathsf{neg}}$ = `GetQueryTuples`($\mathbf{q}$, $\mathtt{F_s}$)
10: ▷ all tuples satisfying $\mathbf{q} \in F_s$ (pos), $\neg\mathbf{q} \in F_s$, (neg)
11: **for every** $\mathbf{t}$ **in** $T_{\mathbf{q}}^{\mathsf{pos}}$ **do**
12: $\mathcal{N}$ = `GenerateNeighborhoods`($\mathbf{t}$, r, k, w)
13: ▷ generate w neighborhoods of depth r and size k
14: **for every** φ **in** Φ **do**
15: $\boldsymbol{\theta} = \varphi/\mathbf{t}$ ▷ substitute query tuple $\mathbf{t}$ in feature φ
16: $x_{\mathbf{t}}^{\varphi}$ = `Count`($\boldsymbol{\theta}$, $\mathcal{N}$, G_{pos})
17: ▷ count groundings satisfied in the neighborhoods
18: **end for**
19: $\mathbf{x}_{\mathbf{t}}^{\mathsf{pos}} = [\ldots, x_{\mathbf{t}}^{\varphi}, \ldots, x_{|\Phi|}]$ ▷ embedding for tuple $\mathbf{t}$
20: **end for**
21: **for every** $\mathbf{t}$ **in** $T_{\mathbf{q}}^{\mathsf{neg}}$ **do**
22: $\mathcal{N}$ = `GenerateNeighborhoods`($\mathbf{t}$, r, k, w)
23: ▷ generate w neighborhoods of depth r and size k
24: **for every** φ **in** Φ **do**
25: $\boldsymbol{\theta} = \varphi/\mathbf{t}$ ▷ substitute query tuple $\mathbf{t}$ in feature φ
26: $x_{\mathbf{t}}^{\varphi}$ = `Count`($\boldsymbol{\theta}$, $\mathcal{N}$, G_{neg})
27: ▷ count groundings satisfied in the neighborhoods
28: **end for**
29: $\mathbf{x}_{\mathbf{t}}^{\mathsf{neg}} = [\ldots, x_{\mathbf{t}}^{\varphi}, \ldots, x_{|\Phi|}]$ ▷ embedding for tuple $\mathbf{t}$
30: **end for**
31: **return** $\mathcal{F} = \{\mathbf{x}_{\mathbf{t}}^{\mathsf{pos}}\}, \{\mathbf{x}_{\mathbf{t}}^{\mathsf{neg}}\}$ ▷ return embeddings
32: **end function**

Procedure 2 presents our method, LGE, for extracting embeddings. We learn the predicates from one of the three rule learning methods, which are then grounded using both the positive and negative examples [**Line 7**]; in addition, tuples of the positive and negative examples are also obtained [**Line 9**]. That is, the positive example `Interacts`(`D13`, `D214`) is described by the positive tuple with two entity arguments: ⟨`D13`, `D214`⟩. Similarly, the negative example ¬`Interacts`(`D149`,`D214`) is described by the negative tuple: ⟨`D149`, `D214`⟩. For both positive ($T_{\mathbf{q}}^{\mathsf{pos}}$) and negative tuples ($T_{\mathbf{q}}^{\mathsf{pos}}$), the neighborhood of each entity in the tuple is obtained, and each predicate is partially grounded with the entities $\in \mathbf{t}$ [**Lines 12-16, 22-26**].

`GenerateNeighborhoods` generates entity neighborhoods for a tuple $\mathbf{t} \in T_{\mathbf{q}}$. Neighborhood generation relies on three parameters: (1) r, the depth of neighborhood when counting, (2) k, the number of neighbors to sample, and (3) w, the number of neighborhoods to be generated. For each entity in tuple $\mathbf{t}$, all neighbors at a maximum distance of r form the neighborhood. This process is repeated until we obtain w neighborhoods for each training example. For example, if $r = 1, w = 5$ and $k = 10$ and we have 10 predicates ($|\Phi| = 10$), we obtain 50 propositional examples with 10 features by looking at 1-neighbors for each entity. The `Count` function **[Lines 16, 26]** counts how many entities in the neighborhood of each query satisfy the partially-grounded predicates. Each such count becomes a propositional feature. In this manner, *we can construct a propositionalized data set of* $|pos| \times w$ *positive examples and* $|neg| \times w$ *negative examples.*

Q2: Does choice of the discriminative algorithm impact the performance?
[Q] Effect of choice of discriminative algorithms: After generating the propositional features from the learned relational rules, we can use any discriminative algorithm for the final prediction. We make use of gradient boosting (GB) and logistic regression (LR) and note that the choice of classifier does not result in significant differences in performance (while using relOCC) after learning relational rules, though the performance of GB is almost always higher than LR. We further conducted experiments with more discriminative machine learning algorithms to show that our method is truly algorithm agnostic. Table 1 shows the result of applying these different machine learning algorithms on propositional features learned by counting over the satisfied relOCC rules for the DDI data set. The performance of all discriminative algorithms is very similar to each other across metrics and across data sets, although the gradient boosting algorithm does outperform the others by small margins thereby answering the research question

Table 1: Effect of choice of discriminative machine learning algorithm for propositional features learned for DDI using the different rule learning methods RWs, ILP and relOCC.

Method	Machine Learning Algorithm	Accuracy	Recall	F1	AUC-ROC	AUC-PR
Random Walk	SVM (linear kernel)	0.653	0.436	0.543	0.641	0.580
	Random Forest (100 estimators)	0.658	0.404	0.528	0.645	0.589
	AdaBoost (300 estimators)	0.664	0.505	0.587	0.656	0.588
	Neural Network (hidden layer size = 1000)	**0.676**	0.481	0.584	**0.666**	**0.604**
	Neural Network (hidden layer size = 5000)	0.675	0.482	0.583	0.665	0.602
	Logistic Regression (with L2 regularization)	0.657	0.469	0.564	0.647	0.581
	Gradient Boosting (300 estimators)	0.669	**0.530**	**0.602**	0.662	0.593
ILP	SVM (linear kernel)	0.682	0.425	0.558	0.668	0.617
	Random Forest (100 estimators)	0.704	0.541	0.634	0.696	0.631
	AdaBoost (300 estimators)	0.732	0.594	0.677	0.725	0.659
	Neural Network (hidden layer size = 1000)	0.807	0.669	0.766	0.800	0.757
	Neural Network (hidden layer size = 5000)	**0.812**	**0.697**	**0.778**	**0.806**	0.757
	Logistic Regression (with L2 regularization)	0.696	0.467	0.592	0.684	0.710
	Gradient Boosting (300 estimators)	0.774	0.674	0.729	0.767	**0.765**
relOCC	SVM (linear kernel)	0.846	0.951	0.854	0.852	0.760
	Random Forest (100 estimators)	0.845	0.813	0.829	0.843	0.788
	AdaBoost (300 estimators)	0.882	0.974	0.886	0.887	0.804
	Neural Network (hidden layer size = 1000)	0.894	0.983	0.897	0.898	0.819
	Neural Network (hidden layer size = 5000)	0.892	0.982	0.896	0.897	0.817
	Logistic Regression (with L2 regularization)	0.860	0.939	0.864	0.864	0.797
	Gradient Boosting (300 estimators)	**0.897**	**0.991**	**0.901**	**0.902**	**0.853**