Machine Learning (2025) 114:123
https://doi.org/10.1007/s10994-025-06753-w

Neural RELAGGS

Lukas Pensel[1] · **Stefan Kramer**[1]

Received: 2 December 2020 / Revised: 22 August 2024 / Accepted: 3 February 2025 /
Published online: 20 March 2025

Abstract
Multi-relational databases are the basis of most consolidated data collections in science and industry today. Most learning and mining algorithms, however, require data to be represented in a propositional form. While there is a variety of specialized machine learning algorithms that can operate directly on multi-relational data sets, propositionalization algorithms transform multi-relational databases into propositional data sets, thereby allowing the application of traditional machine learning and data mining algorithms without their modification. One prominent propositionalization algorithm is RELAGGS by Krogel and Wrobel, which transforms the data by nested aggregations. We propose a new neural network based algorithm in the spirit of RELAGGS that employs trainable composite aggregate functions instead of the static aggregate functions used in the original approach. In this way, we can jointly train the propositionalization with the prediction model, or, alternatively, use the learned aggegrations as embeddings in other algorithms. We demonstrate the increased predictive performance by comparing N-RELAGGS with RELAGGS and multiple other state-of-the-art algorithms.

Keywords Network architecture · Propositionalization · Relational data mining · Aggregation

1 Introduction

While neural networks are among the most used and successful machine learning algorithms at the time, they mostly rely on propositional, i.e. single table, input data. While the support for many types of structured data, such as graphs (Wang et al., 2017; Wu et al., 2020), increased over the last years, only a few approaches consider multi-relational data. Unfortunately, many real-world problems are based on relational data structures, such as social networks, customer accounts, and health records. There are some approaches to

Editor: Nikos Katzouris.

✉ Lukas Pensel
pensel@uni-mainz.de

Stefan Kramer
kramer@informatik.uni-mainz.de

[1] Institut für Informatik, Staudingerweg 9, 55099 Mainz, Germany

incorporate those relational structures into the topology of the neural network, such as relational neural networks (Šourek et al., 2017), and there are many algorithms to transform a relational data set into a propositional data set, such as RELAGGS (Krogel & Wrobel, 2001), in order to make it usable for all traditional machine learning and data mining algorithms.

The goal of this work is to merge these approaches and build a propositionalization algorithm into the topology of a neural network. We aim to combine the simplicity of using propositional data as input for our predictions with the possibility to adjust the way in which the data is transformed, in order to optimize the propositionalization for the specific prediction task while also training the predictor.

Along those lines, we introduce an approach to trainable aggregate functions based on the composition of parameterized mappings and unparameterized aggregate functions. We show further that we can use those functions to jointly propositionalize and predict relational data in a neural network context and we provide an implementation of such an approach with our Neural RELAGGS (N-RELAGGS) algorithm. Lastly, we evaluated our algorithm and compared it with other state-of-the-art static propositionalization methods on multiple benchmark data sets of varying size.

The main contributions of this work are:

- We introduce aggregate functions for propositionalization, in the style of RELAGGS, that are *trainable* and can thus be further optimized in learning. The trainable aggregate functions are composite functions and consist of aggregation layers in neural networks.
- We implement those trainable aggregate functions and provide them in a joint propositionalization and prediction algorithm called N-RELAGGS (Neural RELAGGS). N-RELAGGS is provided with an interface to the python-rdm[1] library and can thus be compared and combined with other algorithms there.
- Experimental results show that N-RELAGGS compares favorably to the compared algorithms, in particular on larger data sets, and that it has a significantly higher prediction performance than most compared algorithms.
- We present a larger dataset derived from the DBLP-Citation-network as a novel relational database benchmark for propositionalization and relational learning methods. Experiments show that N-RELAGGS outperforms RELAGGS on this database as well.

This paper is organized as follows: In Sect. 2 we give a short introduction to relational data mining, introduce the approach of propositionalization and present the algorithms used for comparison in this work. The idea of composite aggregate functions and our implementation of them using neural networks is described in Sect. 3. The experimental setup and the data sets we use are specified in Sect. 4 together with the achieved predictive performance. In Sect. 5 we discuss our results and we present our conclusions and an outlook on further work in Sect. 6.

[1] https://github.com/xflows/rdm/.

2 Related work

In this section we will give a short overview of the general problem of relational data mining and introduce the propositionalization approach to relational data mining. Additionally, we introduce the algorithms used for comparison in our experiments.

2.1 Relational data mining

Relational data mining describes the task of finding patterns among multiple connected data tables, instead of just one single table, as with traditional data mining. Therefore, traditional algorithms cannot be directly applied to this task, which creates the necessity of specialized algorithms and methods to handle relational data. One class of such algorithms are propositionalization algorithms, which are the focus of this work.

2.2 Propositionalization

Propositionalization is the representation change of transforming a relational representation of a learning problem into a propositional (feature-based, attribute-value) representation (Kramer et al., 2001). The idea is to build higher-level feature representations from lower-level relational data, just like super-pixels are constructed from pixels in image data. Taking such an approach, feature construction can be decoupled from model construction. In propositionalization, the search space of relational features is not gradually searched and expanded as in inductive logic programming (ILP), but all relational features with certain properties, e.g., up to some maximal syntactic size or within a minimal and maximal frequency of occurrence, are generated and used to transform the representation. The advantage is that it is possible to take advantage of any progress with propositional learning algorithms in this way. The disadvantage is the potential loss of information due to size or frequency constraints. If propositionalization does not give the desired results, it should at least be used as a baseline to show that more advanced search and optimization strategies are worth the effort.

Propositionalization schemes can be categorized according to the types of features that are constructed. One basic distinction is the one between existential features and aggregate features (see Table 1). Existential features are defined by conjunctive queries, which, when succeeding for an instance, give the value true, and false, otherwise. Variants are possible by counting the number of successful proofs. Aggregate features (Krogel & Wrobel, 2001) are more complex: We consider a defined set of variables except the key, which give the

Table 1 Propositionalization: (a) A conjunctive query with key K defining a Boolean feature for an instance K (true if it succeeds for some K, false otherwise). This is called an existential feature. (b) A clause defining a feature for a relational example.

?	–	person(K), parent(K, Y), has_pet(Y, cat)	(a)
p(K, Z)	:-	person(K), has_account(K, Y), overdraft(Y, Z)	(b)

K denotes the key (i.e., the identifier of the instance). It is assumed that a person may have several accounts Y, each with a different numerical overdraft limit Z. Such a clause may be the basis for several types of features, e.g., by applying aggregate functions like the sum, mean, minimum, or maximum, to the set of different values for Z. A resulting feature is called an aggregate feature

answer substitutions for that query (when the key variable is bound to some instance identifier). In the above example (see Table 1 (b)), K is the key, and the user has defined variable Z to be the one of interest for relational feature construction. When the query is evaluated for some instance K, we gather all values for Z, and, in the final step, apply a user-defined set of aggregate functions (like minimum, maximum, mean, standard deviation, mode, etc.) to that set of values to define propositional values. Clearly, variants are possible: The clause in Table 1 (b) can be the basis for some test against a threshold (e.g., $Z > 10000$), or it can be used not to turn the problem into a propositional problem, but, without aggregates, into a multi-instance learning or multi-tuple learning problem (De Raedt, 2008).

Recent years have seen the rise of so-called *dynamic propositionalization* schemes. Unlike static propositionalization, which execute a static transformation scheme, dynamic propositionalization algorithms, such as Schouterden et al. (2019), construct the features of the propositional representation in a lazy manner that is guided by the learner. Similar to the way how ILP systems gradually construct longer clauses, dynamic propositionalization algorithms gradually expand the feature table. So, while static propositionalization is the largest category of existing approaches, other (dynamic) approaches exist that do not generate the whole feature space at once defined by their language bias. As dynamic propositionalization is located somewhere between propositionalization and relational learning and it does not follow a static transformation scheme, the method proposed in this paper, N-RELAGGS, does not belong to this family of methods.

2.2.1 RELAGGS

The RELAGGS, or relational aggregation, algorithm (Krogel & Wrobel, 2001) is built on aggregate features as described above and transforms a relational data representation into a propositional one by nested aggregation. Given an entry v of a relational data table and corresponding entries W of a related table, RELAGGS transforms v and W into $v \oplus max(W) \oplus \cdots \oplus std(W)$, where $\oplus$,[2] denotes the concatenation of two vectors and $f(W)$ denotes the element-wise application of the aggregate function f (max / std / etc.) resulting in a single vector of aggregates. By iteratively aggregating connected entries of related tables, the algorithm converts a relational database into a propositional data table. Commonly used aggregation functions for this algorithm are average, maximum, minimum, standard deviation, and sum. Since this propositionalization approach is quite simple, we chose to build our approach based upon its structure.

2.2.2 Other comparison algorithms

Aleph One of the most popular ILP systems is Srinivasan (2001). Since Aleph is a relational learner, it generates a set of clauses, given background knowledge, positive examples and negative examples. Those clauses can be turned into definitions of features for classification algorithms, and thus, Aleph can also be used as a propositionalization algorithm.

RSD In order to discover relational subgroups, the RSD algorithm (Železný and Lavrač, 2006) generates a propositional representation of the provided relational database in a first step. This representation can be directly used with other algorithms.

[2] Following the notation of the RELAGGS paper (Krogel & Wrobel, 2001) we use $\oplus$ to denote concatenations of tuples, vectors or tensors.

TreeLiker The ReIF algorithm (Kuželka and Železný, 2011), part of the TreeLiker software, generates a propositional data representation of a relational database, by constructing a set of tree-like features.

Wordification Wordification (Perovšek et al., 2013) is an approach to propositionalization based on text mining methods. It transforms a relational database into a representation akin to a Bag-Of-Words representation.

*PropDRM/PropStar*Lavrač et al. (2020) are two algorithms which combine propositionalization with embedding. PropDRM is an instance-based embedding approach, based on Deep Relational Machines (DRM) and PropStar is a feature-based embedding approach, based on the StarSpace entity embedding.

Aggregate Cascade-Correlation Networks Aggregate Cascade-Correlation Networks (ACCN) (Uwents & Blockeel, 2008) are a group of Cascade-Correlation Networks adapted for multi-relational data. By adding Aggregation Units, the networks are able to utilize bags of vectors as inputs. ACCNs are rather direct predictors for relational data than propositionalization algorithms. Since ACCNs are not propositionalization methods, we do not run experiments with them in this work.

Relational Neural Networks (RelNN) and Graph Neural Networks (GNN) RelNNs and GNNs are two network architectures that predict targets for relational data sets, by utilizing a graph representation of the database (Uwents et al., 2011). The nodes represent the rows of a table and the edges represent the relationships between different rows of different tables. Similar to the ACCNs, RelNNs and GNNs are predictors for relational data rather than propositionalization algorithms.

3 Methods

Here we introduce the idea of composite aggregate functions as trainable aggregate functions and present a possible implementation of those in the form of Neural RELAGGS (N-RELAGGS).

3.1 Learned aggregation

Preset aggregate functions are strongly limited in the amount of information they can retain. Additionally, those functions do not have trainable parameters to fit the aggregate to a specific task. In order to improve the amount of important information for a specific task, we want to construct trainable aggregate functions.

Definition 1 (Aggregate function) An aggregate function ψ maps a bag $\mathcal{B}^I$ of elements of an input domain I to a single element of an output domain O.

$$\psi : \mathcal{B}^I \rightarrow O$$

Definition 2 (Trainable aggregate function) A trainable aggregate function ψ_θ is an aggregate function for each possible set of parameters $\theta \in \Theta$.

$$\psi_\theta : \mathcal{B}^I \rightarrow O$$

Definition 3 (Composite aggregate function) Given a base aggregate function ψ for the input domain I^* and the output domain O^* and two functions f_{θ_1} and g_{θ_2} with

$$f_{\theta_1} : I \to I^*$$
$$g_{\theta_2} : O^* \to O$$

we obtain the composite aggregate function $\psi_{(\theta_1,\theta_2)} = g_{\theta_2} \circ \psi \circ f_{\theta_1}$, which is a trainable aggregate function.

$$\psi_{(\theta_1,\theta_2)} : \mathcal{B}^I \to \mathcal{B}^{I^*} \to O^* \to O$$
$$s \mapsto g_{\theta_2}(\psi(f_{\theta_1}(s))) \qquad s \in \mathcal{B}^I$$

In this work, we integrate the aggregate functions into a neural network, building a composite aggregate function consisting of neural network layers and some base aggregate function. Therefore, we can jointly learn the aggregate function and train the model. An exemplary application of this approach is depicted in Fig. 1. The method design is inspired by global one-dimensional pooling layers, where sequences of data are combined into a single representation. The nomenclature of variables and parameters is gathered in Table 2. In order to use such a global pooling layer to aggregate a batch of n bags $\mathcal{B}_1, \mathcal{B}_2, \dots \mathcal{B}_n$ of data, with bag $\mathcal{B}_i$ consisting of $l_i = |\mathcal{B}_i|$ data entries with m features, we need to transform the data into a three-dimensional tensor with the shape $(n, m, \bar{l} = \sum_{i \leq n} l_i)$. This is achieved by concatenating all collections into a tensor of the shape $(\bar{l}, m, 1)$ and multiplying it with a mask M of the shape $(\bar{l}, 1, n)$, resulting in a tensor of the shape $(\bar{l}, m, n)$. By transposing the tensor, i.e. reversing the order of the axis, we obtain the required shape $(n, m, \bar{l})$. Applying global pooling to this returns a tensor of the shape (n, m). Example 1 shows how this process is applied.

Table 2 Nomenclature of used variables and parameters

Name	Description
n	The number of items in a batch or data set
m	The number of features
$\mathcal{B}$	A bag of data instances
$l = \lvert\mathcal{B}\rvert$	The number of items in a bag of data $\mathcal{B}$
k	The number of base aggregate functions
Λ	A neural network layer
$\lvert\Lambda\rvert$	The number of neurons in the layer Λ
x_{data}	List of data tensors corresponding to the tables of the database
x_{ids}	List of index tensors indicating which data points should be aggregated
P	The order in which the tables of a database are aggregated
p	An entry of P consisting of a list of tables which will be aggregated and a table the aggregations are concatenated with
λ	The relation of the output of a network layer and their input $\lambda = \frac{\lvert\Lambda\rvert}{m}$
$\mathcal{P}$	A sub-tree of the dfs-tree with the target table as root
$\not{\mathcal{P}}_i$	In $\mathcal{P}$ which are $i-1$ steps away the root
$C(\Lambda)$	The number of trainable parameters present in a neural network, i.e. the complexity
$C(\mathcal{P})$	The number of trainable parameters needed to aggregate the data present in $\mathcal{P}$
$C(DB)$	The number of trainable parameters needed to propositionalize the database DB

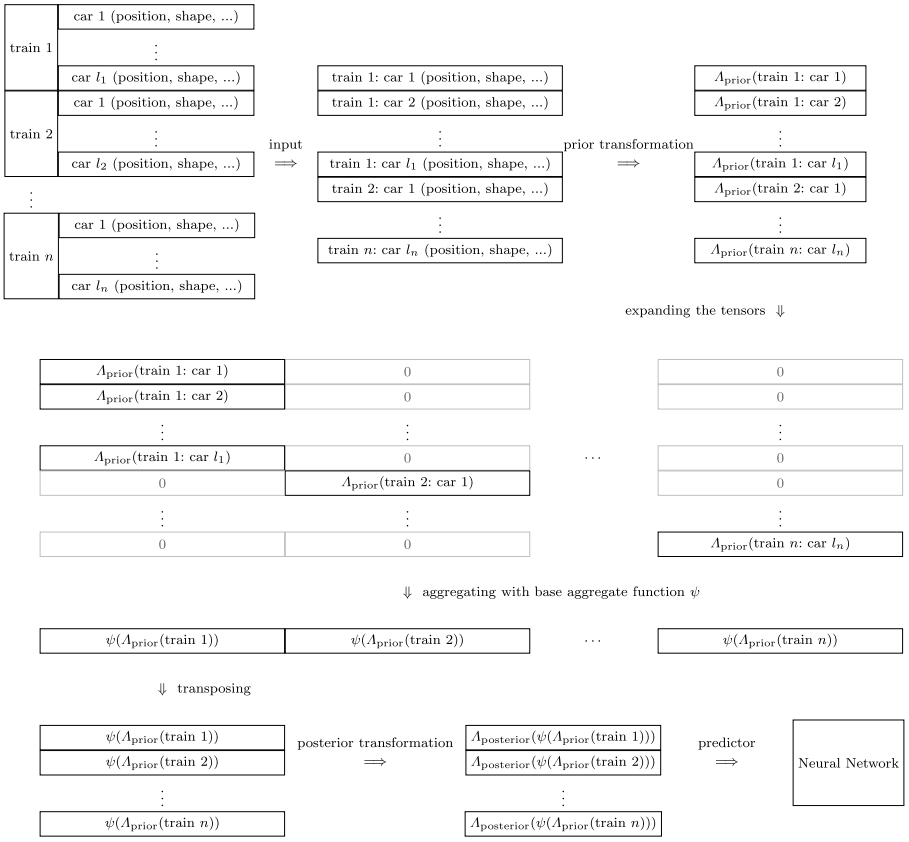


Fig. 1 Schematic of the neural network based composite aggregate function. We take n instances (train 1, train 2,..., train n) with $l_1, \ldots, l_n$ entries as input and each entry is passed through a first neural network layer for the prior transformation. Then the tensors are spanned in order to aggregate the entries of singular instances. The resulting tensor of aggregates is passed through a second neural network layer for the posterior transformation and subsequently forwarded to the predictor model

Example 1 Given the bags $\mathcal{B}_1$ and $\mathcal{B}_2$ with $m = 3$ features

$$\mathcal{B}_1 := \begin{bmatrix} 1 & 2 & 1 \end{bmatrix}, \mathcal{B}_2 := \begin{bmatrix} 3 & 4 & 3 \\ 6 & 5 & 6 \\ 7 & 8 & 7 \end{bmatrix} \Rightarrow n = 2, m = 3, l_1 = 1, l_2 = 3, \bar{l} = 4,$$

we concatenate them into $\mathcal{B}$ and build the corresponding mask M:

$$\mathcal{B} = \mathcal{B}_1 \oplus \mathcal{B}_2 = \begin{bmatrix} [1] & [2] & [1] \\ [3] & [4] & [3] \\ [6] & [5] & [6] \\ [7] & [8] & [7] \end{bmatrix} (\bar{l}, m, 1), M = \begin{bmatrix} \begin{bmatrix} 1 & 0 \end{bmatrix} \\ \begin{bmatrix} 0 & 1 \end{bmatrix} \\ \begin{bmatrix} 0 & 1 \end{bmatrix} \\ \begin{bmatrix} 0 & 1 \end{bmatrix} \end{bmatrix} (\bar{l}, 1, n).$$

To aggregate the entries of the bags, we multiply $\mathcal{B}$ with M, transpose the result and then apply max-pooling to the last dimension.

$$\mathcal{B} \times M = \left[\left[\begin{bmatrix}1 & 0\\ 0 & 3\\ 0 & 6\\ 0 & 7\end{bmatrix} \begin{bmatrix}2 & 0\\ 0 & 4\\ 0 & 5\\ 0 & 8\end{bmatrix} \begin{bmatrix}1 & 0\\ 0 & 3\\ 0 & 6\\ 0 & 7\end{bmatrix}\right]\right] (\bar{l}, m, n)$$

$$(\mathcal{B} \times M)^T = \left[\left[\begin{bmatrix}1 & 0 & 0 & 0\\ 0 & 3 & 6 & 7\end{bmatrix} \begin{bmatrix}2 & 0 & 0 & 0\\ 0 & 4 & 5 & 8\end{bmatrix} \begin{bmatrix}1 & 0 & 0 & 0\\ 0 & 3 & 6 & 7\end{bmatrix}\right]\right] (n, m, \bar{l})$$

$$max(\mathcal{B} \times M)^T = \begin{bmatrix}1 & 2 & 1\\ 7 & 8 & 7\end{bmatrix} (n, m)$$

This results in a tensor of the shape (n, m).

Algorithm 1 Composite neural network aggregation

Data: input data X of shape $(\bar{l}, m, 1)$, input mask M of shape $(\bar{l}, 1, n)$
Result: aggregated data $\tilde{X}$ of shape $(n, \bar{m})$

$X^* = \Lambda_{\text{prior}}(X)$; // $(\bar{l}, m, 1) \to (\bar{l}, m^*, 1)$
$X^* = X^* \times M$; // $(\bar{l}, m^*, 1) \times (\bar{l}, 1, n) \to (\bar{l}, m^*, n)$
$\tilde{X}^* = \text{aggregate}(X^*)$; // $(\bar{l}, m^*, n) \to (m^*, n)$
$\tilde{X}^* = \text{transpose}(\tilde{X}^*)$; // $(m^*, n) \to (n, m^*)$
$\tilde{X} = \Lambda_{\text{posterior}}(\tilde{X}^*)$; // $(n, m^*) \to (n, \bar{m})$

In order to introduce trainable parameters to the aggregate function, we add two dense network layers, as presented in Algorithm 1. In order to aggregate more complex composite features, we transform the input data prior to the aggregation with a dense network layer Λ_{prior} consisting of $|\Lambda_{\text{prior}}|$ neurons, where each input entry with m features is mapped to a processed input entry with $m^* = |\Lambda_{\text{prior}}|$ features. After the aggregation, we transform the data again with a dense network layer $\Lambda_{\text{posterior}}$ consisting of $|\Lambda_{\text{posterior}}|$ neurons, to select the most promising aggregates and build features based on the combination of aggregates. This maps the aggregates of shape (n, m^*) to a selected output of shape $(n, \bar{m})$, with $\bar{m} = |\Lambda_{\text{posterior}}|$.

In this manner, we can use one specific pooling method, such as average or max pooling. However, we can increase the expressiveness of the aggregate by incorporating multiple different pooling, i.e. aggregation, methods and concatenating the results, similar to the RELAGGS algorithm. So, if k is the number of different pooling methods used for aggregation, we get the final sequence of tensors:

$$\underbrace{(\bar{l}, m) \to (\bar{l}, m^*)}_{\text{prior transformation}} \to \underbrace{(\bar{l}, m^*, 1) \times (\bar{l}, 1, n) \to \overbrace{(\bar{l}, m^*, n)}^{\text{aggregate}}}_{\text{expand the tensors}} \to \underbrace{(n, k \times m^*) \to (n, \bar{m})}_{\text{posterior transformation}}$$

3.2 Neural RELAGGS

Our goal is to transform a multi-relational data set, a relational database in snowflake schema with provided target table and target attribute, by aggregating, using trainable

aggregate functions, into a propositional representation and take this representation as input for a prediction network.

In order to handle a multi-relational data set, we have to combine multiple aggregations into one network. The implementation of those aggregation layers is described in Sect. 3.2.2. Additionally, we have to transform the multi-relational data set into a representation we can feed into a neural network and retrieve the structure of the relations, in order to build it into the network architecture. This transformation and the retrieval of the structure as aggregation plan P is described in Sect. 3.2.1.

3.2.1 Data preprocessing

Algorithm 2 generate_aggregation_plan

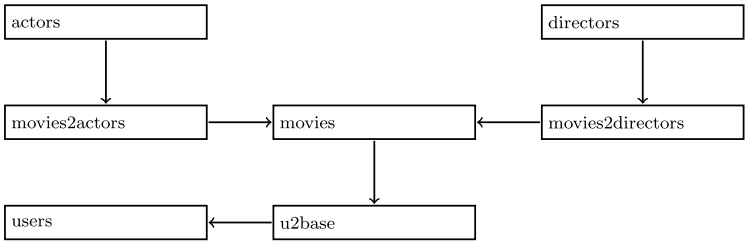
```
Data: database context DB
Result: aggregation plan P
connections = DB.connected;
/* Depth-first search for paths in the database, starting with the
   target table                                                    */
Def path_dfs(current, visited):
    out = ([], current);
    next_out = [];
    forall con ∈ connections do
        old, new = con;
        if old == current then
            if new ∉ visited then
                out[0].append(new);
                next_out = next_out ∪ path_dfs(new, visited ∪ new);
            end
        end
    end
    if |out[0]| > 0 then
        return out ∪ next_out;        // Pass the previously found results on
    else
        return [];                    // No additional aggregations possible
    end
/* Plan starts with the target table                               */
P = path_dfs(DB.target_table, [DB.target_table]);
```

Algorithm 3 database_converter

Data: database context DB
Result: Data X with $x \in X$ represented as $x = [x_{data}, x_{ids}]$, target values Y, aggregation plan P
$DB^* = preprocess(DB)$; `/* preprocess values of tables: scale numeric data and binarize categorial data */`
$P^* = generate\_aggregation\_plan(DB^*)$; `// See Algorithm 2`
$X = []$;
$Y = []$;
forall $data\_instance \in DB^*.target\_table$ **do**
  $Y.append(data\_instance.target)$;
  $x_{data} = [[data\_instance], [], \ldots []]$; `/* The first` x_{data} `entry is the target table */`
  $x_{ids} = [[], [], \ldots []]$;
  forall $nexts, current \in P^*$ **do**
    forall $next \in nexts$ **do**
      $next\_entries = where\_connected(x_{data}[current], DB^*.next)$;
      $x_{data}[next] \mathrel{+}= next\_entries$; `/* Add the connected entries of table` $next$ `*/`
      $x_{ids}[next] \mathrel{+}= next\_entries.identifier$; `/* Store the connections between the data entries */`
    end
  end
  $X.append([x_{data}, x_{ids}])$;
end
$P = invert(P^*)$; `// We need to aggregate towards the target table`

Given a database *DB* in snowflake schema, a target table *DB.target_table* and a target attribute *DB.target_table.target*, we transform *DB* into the data instances *X* and target values *Y* and generate the aggregation plan *P*. As shown in Algorithm 3, we first transform the values of the relational tables into an usable format by scaling and binarizing the table entries. Additionally, embeddings could also be incorporated in this step for more complex raw data like texts. Before the data set is constructed, the aggregation plan P^* is generated by a depth-first search through the database structure presented in Algorithm 2. During this search the connections between tables are memorized in tuples $p \in P^*$ with $p = (nexts, current)$, where *current* is the description of the currently visited table and *nexts* is a list of previously unvisited tables connected to the current table. After the data transformation, we get the final aggregation plan *P* by inverting the order of P^*, so that the tables farthest away from the target table get aggregated first and all data entries collapse into a single feature vector. The relation of the aggregation plan to the database structure is depicted in Fig. 2.

Then, for each entry in the target table, an instance in the output data set is generated. Here the target attribute is saved as *y* and added to *Y* and the database is traversed according to P^*. For all entries of the tables in *nexts* that are connected to the previously selected entries of the current table, those entries are selected into the relation of the data instance and an identifier is stored to correctly map the entries for aggregation. After a complete pass through P^*, the instance $x = [x_{data}, x_{ids}]$ is added to the processed data set *X*. Since the processing of individual data instances is independent from one another, this process is highly parallelizable.



$\Downarrow$ Algorithm 2

$P^* =$ [([u2base], users), ([movies], u2base), ([movies2actors, movies2directors], movies), ([directors], movies2directors), ([actors], movie2actors)]

$\Downarrow$ Invert

$P =$ [([actors], movie2actors), ([directors], movies2directors), ([movies2actors, movies2directors], movies), ([movies], u2base), ([u2base], users)]

Fig. 2 Relation of the aggregation plan P for the MovieLens database

Our data processing approach is based on the database connection and context capabilities of the python-rdm package and therefore easy to deploy along other algorithms and functionalities provided by that package.

3.2.2 Aggregation layer

Since there are only global average pooling and global max pooling layers implemented in TensorFlow (Abadi et al., 2015), we would be limited to just use those two base aggregate functions. However, internally the pooling layers are implemented using dimension reducing operations, such as sum or mean. Therefore, we can employ more base aggregate functions by incorporating those operations directly instead of the pooling layers. As described above, we first transform the input into a tensor of the shape $(\bar{l}, m^*, n)$ by generating features and expanding it using mask M. Now the tensor is reduced along the first dimension, resulting in a tensor of shape (m^*, n) and after transposing (n, m^*).

Unfortunately, this approach requires us to construct huge tensors in the expanding step, which cannot be stored in the systems memory for bigger data sets. Theoretically, the use of sparse matrices should eliminate this problem, but to our knowledge there exists no available implementation of genuine sparse tensor multiplication, where the result is a sparse tensor and internally everything is sparsely represented and not cast to a dense tensor.

To solve this problem, we incorporate the idea of using the TensorFlow segment operations as base aggregate functions.[3] Given a two-dimensional tensor X of shape $(\bar{l}, m^*)$ and $\bar{l}$ corresponding indices J, with $j <= n \forall j \in J$, relating each entry of X to its data instance, the segment operation reduces X along the first axis to a tensor of shape (n, m^*), by

[3] Michael Larionov. "Learning Aggregate Functions." Medium, 4 February 2019, towardsdatascience.com/learning-aggregate-functions-5b4102383433. Accessed 14 August 2024.

combining all entries with the same index. Instead of computing an $(\bar{l}, m, n)$ tensor, we only need an $(\bar{l}, m^*)$ and an $(\bar{l}, 1)$ tensor. This alters the sequence of tensors to:

$$\overbrace{(\bar{l}, m) \rightarrow \underbrace{(\bar{l}, m^*)}_{\text{combine data with indices}}}^{\text{prior transformation}} \rightarrow \overbrace{\underbrace{(\bar{l}, m^*), (\bar{l}, 1)}_{\text{combine data with indices}} \rightarrow \underbrace{(n, k \times m^*)}_{\text{posterior transformation}}}^{\text{aggregate}} \rightarrow (n, \bar{m}).$$

Since we incorporate the sum, maximum, minimum and average as base aggregates in our implementation, we can learn them by setting a neuron to pass through one unmodified value. Additionally, we can learn those base aggregates of combinations of features. For instance, the maximum, minimum, average or total difference between two features is calculated by assigning one feature the weight of 1, the other of -1 and all other features of 0. However, aggregate functions which require knowledge about the whole collection of data during the feature generation step, such as the standard deviation, cannot be exactly recreated.

3.2.3 N-RELAGGS

With the preprocessing of the data described in Sect. 3.2.1 and the aggregation layer described in Sect. 3.2.2 we can build the complete N-RELAGGS algorithm.

The transformed data X can now be aggregated according to the aggregation plan P, and the resulting propositional data representation is then fed into a prediction network Λ_{pred}, as described in Algorithm 4. Since the whole network architecture is jointly implemented, it also can be jointly trained using standard stochastic gradient descent and specific optimizers like Kingma and Ba (2014).

To minimize the number of hyperparameters, we just set a prior transformation factor λ_{prior} and a posterior transformation factor $\lambda_{\text{posterior}}$ to adjust the aggregation layers. They describe the factor by which the number of features is multiplied, i.e. the relation between the input size and the output size of the transformation layers $\lambda_{\text{prior}} = \frac{|\Lambda_{\text{prior}}|}{m}$. A factor of 1.0 retains the number of features after the pass through the network layer, where a factor of 0.5 halves the number of features.

We implemented the N-RELAGGS algorithm in Python using TensorFlow as the underlying deep-learning framework and made the source code publicly available.[4]

[4] https://github.com/kramerlab/n-relaggs.

Algorithm 4 Neural RELAGGS

Data: Relational database in snowflake schema DB, with target table $DB.target\_table$ and target attribute $DB.target\_table.target$
Result: Predicted values y_pred, true targets y_true
$X, y\_true, P = \text{database\_converter}(DB)$; `// see Algorithm 3`
$y\_pred = []$;
forall $x_{data}, x_{ids} \in X$ **do**
  forall $p \in P$ **do**
    $x_{agg} = []$;
    forall $i \in p[0]$ **do**
      $x^* = \Lambda^i_{\text{prior}}(x_{data}[i])$; `// Prior transformation`
      $\tilde{x}^* = (segment\_sum(x^*, x_{ids}[i])) \oplus (segment\_mean(x^*, x_{ids}[i])) \oplus (segment\_min(x^*, x_{ids}[i])) \oplus (segment\_max(x^*, x_{ids}[i]))$;
      `// Aggregate data`
      $\tilde{x} = \Lambda^i_{posterior}(\tilde{x}^*)$; `// Posterior transformation`
      $x_{agg}.append(\tilde{x})$
    end
    forall $i \in p[1]$ **do**
      $x_{agg}.append(x_{data}[i])$; `// Non-aggregated part of the combination`
    end
    $x_{agg} = \bigoplus_{x \in x_{agg}} x$;
    $x_{data}[p[1]] = x_{agg}$; `// Store data for later combination`
  end
  $y\_pred.append(\Lambda_{pred}(x_{data}[0]))$; `// All data is aggregated into` $x_{data}[0]$
end

3.3 Computational complexity

In this section we discuss the computational complexity of the N-RELAGGS algorithms regarding the number of trainable parameters. Since the structure of arbitrary databases can be very complex, we assume that the regarded database *DB* has a snowflake schema and the hyperparameters λ_{prior} and $\lambda_{\text{posterior}}$ both are set to smaller than or equal to 1.0.

Each aggregation layer consists of $\lambda_{\text{prior}} \times m$ neurons in the prior transformation layer and $4 \times \lambda_{\text{posterior}} \times \lambda_{\text{prior}} \times m$ neurons in the posterior transformation layer, with m denoting the number of input features and 4 relating to the 4 different base aggregation functions. Since every neuron in this layer has one trainable bias parameter and one trainable weight for each element in the input tensor, which are m and $4 \times \lambda_{\text{prior}} \times m$ respectively, the total number of trainable parameters $C(\Lambda_{\text{aggregation}}(m))$ for an aggregation layer $\Lambda_{\text{aggregation}}(m)$ with m input features is:

$$C(\Lambda_{\text{aggregation}}(m)) = \underbrace{\overbrace{(m+1)}^{\text{inputs and bias}} \times \overbrace{(\lambda_{\text{prior}} \times m)}^{|\Lambda_{\text{prior}}|}}_{\Lambda_{\text{prior}}} + \underbrace{\overbrace{(4 \times \lambda_{\text{prior}} \times m + 1)}^{\text{inputs and bias}} \times \overbrace{(4 \times \lambda_{\text{posterior}} \times \lambda_{\text{prior}} \times m)}^{|\Lambda_{\text{posterior}}|}}_{\Lambda_{\text{posterior}}}$$

This can be approximated by:

Table 3 Statistics of the different databases used for the experiments. For each table, the number of columns and rows as well as the distribution of the target attribute for the target table are given

Database	Tables	#columns	#rows	Target
Trains	Cars	10	63	Direction: east(10), west(10)
	Trains	2	20	
Mutagenesis 42/188	Atoms	5	1001/4893	Active: 1(13/125), 0(29/63)
	Bonds	5	1066/5243	
	Drugs	7	42/188	
	Rings	2	259/1317	
	ring_atom	3	1785/9310	
	ring_strucs	3	279/1433	
Carcinogenesis	Atom	5	8855	Class: 1(182), 0(147)
	canc	2	329	
	sbond_1	4	13340	
	sbond_2	4	940	
	sbond_3	4	12	
	sbond_7	4	4094	
IMDb top	Actors	4	7118	Quality: 1(122), 0(44)
	Directors	3	130	
	directors_genres	4	1123	
	Movies	4	166	
	movies_directors	3	180	
	movies_genres	3	408	
	roles	4	7738	
Movielens	Actors	3	99129	u_gender: F(1708), M(4331)
	Directors	3	2201	
	movies	5	3832	
	movies2actors	3	152532	
	movies2directors	3	4141	
	u2base	3	946828	
	Users	4	6039	

$$
\begin{aligned}
C(\Lambda_{\text{aggregation}}(m)) &= (m+1) \times (\lambda_{\text{prior}} \times m) + (4 \times \lambda_{\text{prior}} \times m + 1) \times (4 \times \lambda_{\text{posterior}} \times \lambda_{\text{prior}} \times m) \\
&\leq (1 \times m) + (1 \times m^2) + (4 \times 1 \times 1 \times m) + (4^2 \times 1 \times 1 \times m^2) \\
&\leq (5 \times m) + (17 \times m^2) \\
&\leq 22 \times m^2 \\
&\in \mathcal{O}(m^2)
\end{aligned}
$$

In order to propositionalize a relational database DB, we aggregate the tables along connections found by a depth-first search over the foreign key relations without visiting the same table multiple times in a path. This search yields a tree $\mathcal{DB}$ with the nodes representing the tables of the DB, the edges representing foreign key relations between tables and

Table 4 The different hyperparameter values for the N-RELAGGS model, the predictor networks and PropStar and PropDRM

Algorithm	Parameter	Values
N-RELAGGS	λ_{prior}	0.5, 0.75, 1.0
	$\lambda_{\text{posterior}}$	0.5, 0.75, 1.0
Predictor network	Layer sizes	(50,), (100,), (100,50)
PropStar/PropDRM	Learning rate	0.01, 0.001, 0.0001
	Number of features	10000, 30000, 50000
	Hidden size	8, 16, 32

the root of the tree corresponds to the target table. Let $\mathcal{P}$ be a sub-tree of $\mathcal{DB}$ with the target table as root and let $d < |\mathcal{P}|$ be the depth of $\mathcal{P}$. Let $\not{\ }_i \in \mathcal{P}$ with $1 \leq i \leq d$ denote all tables T of $\mathcal{P}$, i.e. the nodes, at level i, i.e. $i-1$ steps away from the root, and let m_T denote the number of features of table T. The total number of trainable parameters of the sub-tree $\mathcal{P}$ is denoted by $C(\mathcal{P})$, and $C(\mathcal{P}, i)$ and $m(\mathcal{P}, i)$ denote the number of parameters for level i and the input size for level i, respectively.

$$
\begin{aligned}
m(\mathcal{P}, i) &\leq \sum_{j=i}^{d} ((4 \times \lambda_{\text{posterior}} \times \lambda_{\text{prior}})^{j-i} \times \sum_{T \in \not{\ }_j} m_T) \\
&\leq (d-i) \times 4^{d-i} \times \sum_{T \in \mathcal{P}} m_T \\
&\leq d \times 4^d \times (|\mathcal{P}| \times \max_{T \in \mathcal{P}} m_T) \\
&\leq |\mathcal{P}|^2 \times 4^{|\mathcal{P}|} \times \max_{T \in \mathcal{P}} m_T \\
C(\mathcal{P}, i) &\leq C(\Lambda_{\text{aggregation}}(m(\mathcal{P}, i))) \\
&\leq 22 \times (|\mathcal{P}|^2 \times 4^{|\mathcal{P}|} \times \max_{T \in \mathcal{P}} m_T)^2 \\
&= 22 \times |\mathcal{P}|^4 \times 16^{|\mathcal{P}|} \times (\max_{T \in \mathcal{P}} m_T)^2 \\
C(\mathcal{P}) &= \sum_{i=1}^{d} C(\mathcal{P}, i) \\
&\leq d \times (22 \times |\mathcal{P}|^4 \times 16^{|\mathcal{P}|} \times (\max_{T \in \mathcal{P}} m_T)^2) \\
&\leq 22 \times |\mathcal{P}|^5 \times 16^{|\mathcal{P}|} \times (\max_{T \in \mathcal{P}} m_T)^2 \\
&\in \mathcal{O}(|\mathcal{P}|^5 \times 16^{|\mathcal{P}|} \times (\max_{T \in \mathcal{P}} m_T)^2)
\end{aligned}
$$

Let $|DB|$ denote the number of tables in DB and $h \leq |DB|$ the number of distinct sub-trees with the target table as root $\mathcal{P}_1, \ldots, \mathcal{P}_h$ with $|\mathcal{P}_i| \leq |DB|^2$ for $1 \leq i \leq h$.

Table 5 Classification accuracy for the different algorithms and databases. For each algorithm and data pair, the average accuracy and the standard deviation over the two ten-fold cross-validations are reported. Results marked with - are omitted because the run-time exceeded 24 h

Algorithm	Trains	Mutagenesis 188	Mutagenesis 42	Carcinogenesis	IMDb Top	Movielens
Majority vote	0.50 (0.00)	0.665 (0.023)	0.695 (0.088)	0.553 (0.012)	0.735 (0.025)	0.717 (0.017)
Aleph	0.60 (0.37)	0.744 (0.090)	0.785 (0.192)	0.477 (0.024)	0.561 (0.120)	–
RSD	0.55 (0.27)	0.846 (0.073)	**0.820 (0.173)**	0.578 (0.088)	0.699 (0.073)	–
Treeliker	0.65 (0.32)	0.862 (0.078)	0.785 (0.156)	0.558 (0.066)	0.813 (0.091)	–
Wordification	0.42 (0.33)	0.806 (0.091)	0.705 (0.205)	0.597 (0.082)	0.607 (0.130)	0.717 (0.000)
PropStar	0.62 (0.31)	0.869 (0.080)	0.690 (0.216)	0.581 (0.077)	0.825 (0.096)	0.716 (0.022)
PropDRM	0.53 (0.19)	**0.890 (0.060)**	0.728 (0.167)	**0.603 (0.078)**	0.799 (0.065)	0.719 (0.024)
RELAGGS	**0.70 (0.24)**	0.840 (0.072)	0.725 (0.211)	0.570 (0.074)	**0.853 (0.080)**	0.711 (0.066)
Fix N-RELAGGS	0.62 (0.31)	0.886 (0.063)	0.748 (0.253)	0.593 (0.069)	0.833 (0.086)	0.743 (0.032)
N-RELAGGS	0.60 (0.25)	0.867 (0.068)	0.808 (0.217)	0.577 (0.063)	0.813 (0.095)	**0.750 (0.035)**

The best results are highlighted in bold

Table 6 AUROC scores for the different algorithms and databases. The reported values are the average and the standard deviation over the two ten-fold cross-validations. Results marked with - are omitted because the run-time exceeded 24 h

Algorithm	Trains	Mutagenesis 188	Mutagenesis 42	Carcinogenesis	IMDb Top	Movielens
Aleph	0.68 (0.43)	0.803 (0.074)	**0.787 (0.239)**	0.533 (0.034)	0.572 (0.157)	–
RSD	0.55 (0.50)	0.946 (0.043)	0.738 (0.349)	0.605 (0.085)	0.839 (0.087)	–
Treeliker	0.65 (0.48)	0.918 (0.061)	0.746 (0.326)	0.589 (0.075)	0.870 (0.092)	–
Wordification	0.55 (0.50)	0.861 (0.076)	0.583 (0.344)	0.642 (0.087)	0.726 (0.128)	0.544 (0.024)
PropStar	0.45 (0.44)	0.851 (0.103)	0.263 (0.280)	0.597 (0.105)	0.688 (0.202)	0.531 (0.045)
PropDRM	**0.70 (0.40)**	0.933 (0.057)	0.775 (0.225)	**0.666 (0.074)**	0.826 (0.179)	0.531 (0.034)
RELAGGS	**0.70 (0.46)**	0.936 (0.050)	0.629 (0.433)	0.614 (0.068)	**0.893 (0.097)**	0.667 (0.120)
Fix N-RELAGGS	0.65 (0.48)	**0.959 (0.052)**	0.758 (0.317)	0.622 (0.064)	0.884 (0.103)	**0.720 (0.043)**
N-RELAGGS	0.65 (0.48)	0.951 (0.038)	0.779 (0.308)	0.619 (0.083)	0.879 (0.112)	0.711 (0.070)

The best results are highlighted in bold

Table 7 Average ranks of the algorithms based on the performance in accuracy, AUROC and the combination of both

Accuracy		AUROC		Combined	
Algorithm	Rank	Algorithm	Rank	Algorithm	Rank
Fix N-RELAGGS	2.62	Fix N-RELAGGS	2.58	Fix N-RELAGGS	2.60
N-RELAGGS	3.62	N-RELAGGS	3.04	N-RELAGGS	3.33
RELAGGS	4.58	PropDRM	3.67	PropDRM	4.19
PropDRM	4.71	RELAGGS	3.92	RELAGGS	4.25
Treeliker	5.33	Treeliker	5.50	Treeliker	5.42
PropStar	5.38	RSD	5.92	RSD	6.04
RSD	6.17	Wordification	6.12	Wordification	6.48
Wordification	6.83	Aleph	7.42	PropStar	6.48
Majority vote	7.67	PropStar	7.58	Aleph	7.75
Aleph	8.08	Majority vote	9.25	Majority vote	8.46

$$\begin{aligned} C(DB) &= \sum_{i=1}^{h} C(\mathcal{P}_i) \\ &\leq |DB| \times 22 \times (|DB|^2)^5 \times 16^{|DB|^2} \times (\max_{T \in DB} m_T)^2 \\ &\in \mathcal{O}(|DB|^{11} \times 16^{|DB|^2} \times (\max_{T \in DB} m_T)^2) \end{aligned}$$

Therefore, the number of trainable parameters is governed by an exponential function in the number of tables of the database in the case of $\lambda_{\text{prior}} \leq 1.0$ and $\lambda_{\text{posterior}} \leq 1.0$.

If we instead assume $\lambda_{\text{prior}} \times \lambda_{\text{posterior}} \leq 0.25$, which is the case, e.g., for $\lambda_{\text{prior}} \leq 0.5$ and $\lambda_{\text{posterior}} \leq 0.5$, we obtain a way better result. Since $(4 \times \lambda_{\text{posterior}} \times \lambda_{\text{prior}}) \leq 1.0$ for $\lambda_{\text{prior}} \times \lambda_{\text{posterior}} \leq 0.25$, we get the following complexity:

$$\begin{aligned} l(\mathcal{P}, i) &\leq |\mathcal{P}|^2 \times 1^{|\mathcal{P}|} \times \max_{T \in \mathcal{P}} m_T = |\mathcal{P}|^2 \times \max_{T \in \mathcal{P}} m_T \\ C(\mathcal{P}, i) &\leq 22 \times |\mathcal{P}|^4 \times (\max_{T \in \mathcal{P}} m_T)^2 \\ C(\mathcal{P}) &\leq 22 \times |\mathcal{P}|^5 \times (\max_{T \in \mathcal{P}} m_T)^2 \\ &\in \mathcal{O}(|\mathcal{P}|^5 \times (\max_{T \in \mathcal{P}} m_T)^2) \\ C(DB) &\leq |DB| \times 22 \times |DB|^{10} \times (\max_{T \in DB} m_T)^2 \\ &\in \mathcal{O}(|DB|^{11} \times (\max_{T \in DB} m_T)^2) \end{aligned}$$

So, the number of trainable parameters is no longer exponential in the number of tables of the database, if we assume $\lambda_{\text{prior}} \times \lambda_{\text{posterior}} \leq 0.25$.

Table 8 Statistical significant differences of the algorithms based on their ranks for the performance measures

Accuracy			AUROC			Combined		
N-RELAGGS	>	Aleph	RELAGGS	>	Majority vote	RELAGGS	>	Aleph
N-RELAGGS	>	Majority vote	N-RELAGGS	>	Aleph	RELAGGS	>	Majority vote
Fix N-RELAGGS	>	Aleph	N-RELAGGS	>	PropStar	N-RELAGGS	>	Aleph
Fix N-RELAGGS	>	Wordification	N-RELAGGS	>	Majority vote	N-RELAGGS	>	Wordification
Fix N-RELAGGS	>	Majority vote	Fix N-RELAGGS	>	Aleph	N-RELAGGS	>	PropStar
			Fix N-RELAGGS	>	PropStar	N-RELAGGS	>	Majority vote
			Fix N-RELAGGS	>	Majority vote	Fix N-RELAGGS	>	Aleph
			PropDRM	>	PropStar	Fix N-RELAGGS	>	RSD
			PropDRM	>	Majority vote	Fix N-RELAGGS	>	Treeliker
						Fix N-RELAGGS	>	Wordification
						Fix N-RELAGGS	>	PropStar
						Fix N-RELAGGS	>	Majority vote
						PropDRM	>	Aleph
						Treeliker	>	Majority vote
						PropDRM	>	Majority vote

We use the Bonferroni-Dunn test with $\alpha = 0.05$ to test the significance

3.4 Feature extraction

After we have trained an N-RELAGGS model, it is able to transform a multi-relational data instance into a single predicted value or, by extracting an intermediate layer of the model, into a fixed size tensor representation. Those representations can be seen as an embedding of multi-relational data into a space of fixed dimension and therefore can be used to propositionalize the relational data and use the transformed data with arbitrary propositional learning algorithms.

4 Experiments

In this section we present the data sets used in our experiments as well as the performance we could measure on those. We test our proposed N-RELAGGS algorithm as well as seven other algorithms on multiple data sets of varying sizes and analyze if there are significant differences in the predictive performance of the algorithms.

Since most commonly used benchmark data sets for relational data mining are rather small, we constructed an additional data set based on the dblp citation network. The data set and the experiments conducted on it are presented in Sect. 4.4.

4.1 Data

The statistics for the different databases are presented in Table 3. For each database, all contained tables with their number of columns and rows are given. Additionally, for the target tables, the target attribute as well as the distribution of the different classes are specified.

Trains The Trains Michie et al. (1994) data set was constructed for the East–West trains challenge problem. In the challenge the direction of each train has to be predicted, based on the properties of the cars. Each train contains a variable number of cars and each has a shape and carries a load. The data set is publicly available in the Relational Dataset Repository.[5]

Mutagenesis The data set (Debnath et al., 1991) comprises of 230 different chemical compounds. The task for this problem is to predict the mutagenicity of the different compounds. The data set is split into two distinct sets, one of 42 and one of 188 instances, and is available in the Relational Dataset Repository.

Carcinogenesis In this task (Srinivasan et al., 1997), the carcinogenicity of different chemical compounds has to be predicted, based on the supplied structure of the compounds. This data set is also publicly available in the Relational Dataset Repository.

IMDb Top The IMDb Top[6] data set is a subset of the internet movie database (IMDb) with the goal to classify the 166 selected movies either as a member of the IMDb top-250 chart or as a member of the IMDb bottom-100 chart.

MovieLens This Khosravi et al. (2012) is another similar database to IMDb, but for this task, the gender of the users has to be predicted. This database is publicly available in the Relational Dataset Repository.

4.2 Setup

The databases are loaded with the python-rdm package and split into train and test sets for two ten-fold stratified cross-validations. Those data sets are then fed into the Python wrappers of the baseline algorithms, provided by the python-rdm package, resulting in different propositional representations of those data sets. Additionally, the relational data structure is converted into numpy arrays, as described in Sect. 3.2.1 and passed to the N-RELAGGS model as well as propositionalized by our own RELAGGS implementation.

A feed-forward neural network of the same structure and with the same set of possible hyperparameters as the predictor part of the N-RELAGGS model is used as predictor for the different propositionalization techniques. The optimal hyperparameters, see Table 4, for each predictor and the N-RELAGGS model are selected by the highest average area under the receiver operating characteristic curve (AUROC) in a stratified three-fold cross-validation over the train set. The neural networks are trained for 100 epochs, with the option of stopping early in the case of stagnating loss improvement, and we use hinge loss and Adam by Kingma and Ba (2014) to optimize the networks. For the comparison algorithms, except for PropStar and PropDRM, all hyperparameters are set to the default values.

[5] https://relational.fit.cvut.cz/.

[6] http://kt.ijs.si/janez_kranjc/ilp_datasets/imdb_top.sql.

Fig. 3 Comparison between the N-RELAGGS algorithm, respectively the Fix N-RELAGGS algorithm, with all other tested algorithms. Green fields show superiority, red fields show inferiority and the saturation of the fields represents the significance of the difference. Additionally, each field yields the p-value determined by a corrected repeated 10-fold cv test with two repetitions and the bar for significance set to $\alpha = 0.05$

In addition to the regular N-RELAGGS, we conduct the experiments with a fixed N-RELAGGS (Fix N-RELAGGS), for which λ_{prior} and $\lambda_{\text{posterior}}$ are fixed to 1.0. This makes Fix N-RELAGGS more comparable to the RELAGGS algorithm.

Since the implementation of the PropStar and PropDRM algorithms has no interface to the python-rdm package, we implemented a method to run the experiments on the same train/test splits. The used hyperparameter space for PropStar and PropDRM is also presented in Table 4.

The experiments are executed on computing nodes with Intel Xeon CPUs, no GPUs and up to 380GB of RAM.

4.3 Results

The experimental results of the selected algorithms on the different data sets are presented in this section. Table 5 yields the measured accuracies and Table 6 the measured AUROC scores for the different algorithm and data pairs. The average ranks of the algorithms based on their performance is presented in Table 7.

Using the ranks of the algorithms, we can test if there are significant differences in the performance of the algorithms. Employing an adapted Friedman test (Demšar, 2006), we can show that there are significant ($p < 0.001$) differences in the distribution of ranks, i.e. the quality of the algorithms. To determine significant differences between the algorithms, we use the Bonferri-Dunn test (Demšar, 2006) with a significance threshold of $\alpha = 0.05$ and present the results in Table 8.

Figure 3 shows for which data sets the N-RELAGGS or Fix N-RELAGGS algorithm performs better or worse than the other algorithms, as well as the significance of the difference indicated by the p-value determined by a corrected repeated 10-fold cv test (Bouckaert & Frank, 2004) with two repetitions.

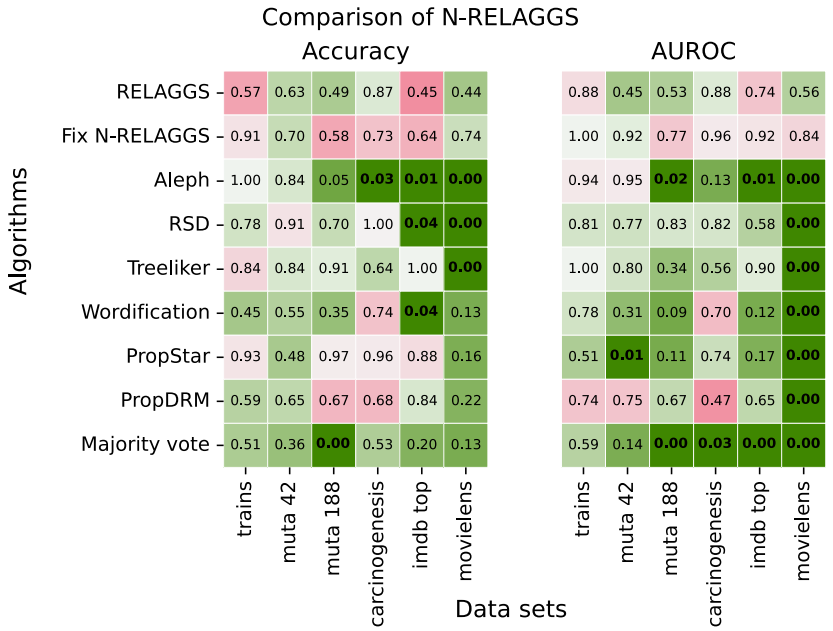
4.4 DBLP data set

This database is constructed from the DBLP-Citation-network V12[7] data set (Tang et al., 2008), in order to build a large benchmark data set. The instances are the papers published between 2010 and 2012 with a threshold for the number of citations used as target. The positive class 1 consists of all papers with more than 10 citations and the negative class 0 of all papers with 10 or fewer citations. In order to prevent leakage, the database table all_references is solely composed of papers published prior to 2010. Table 9 presents the overall statistics of the DBLP database.[8]

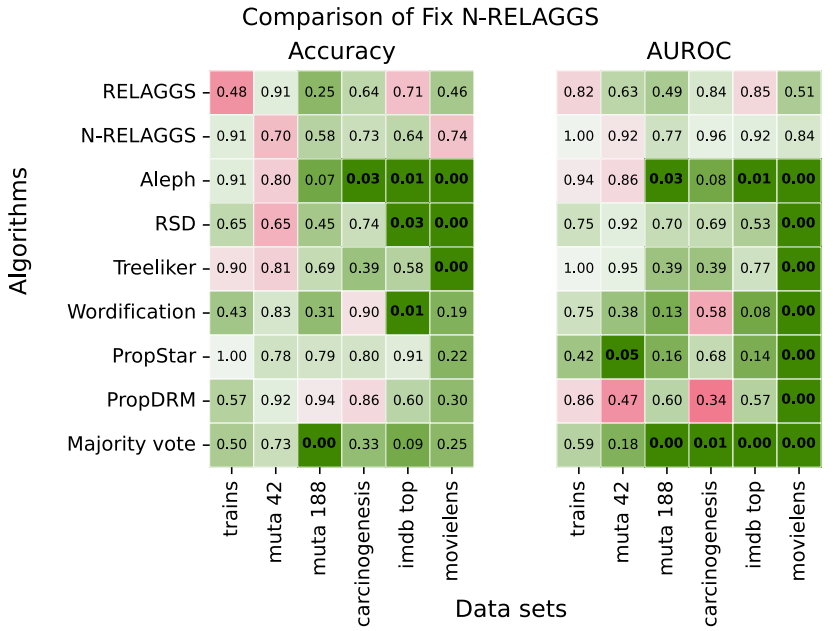
Because of the size of the database, most algorithms are not able to propositionalize the database in 5 days, which is the longest possible run time on the compute nodes. Just our

[7] https://originalstatic.aminer.cn/misc/dblp.v12.7z.

[8] https://github.com/kramerlab/n-relaggs/Data/DBLP.



(a) N-RELAGGS



(b) Fix N-RELAGGS

Table 9 Statistics of the DBLP database used for the experiments

Tables	#columns	#rows	Target
all_references	5	1639164	
author2paper	3	2787326	
org2paper	2	888618	
paper2author	3	1386766	
paper2org	2	473104	
paper2paper	2	5984894	
papers	5	724199	Target: 0(504754), 1(219445)

For each table, the number of columns and rows as well as the distribution of the target attribute for the target table are given

Table 10 Scores for experiments conducted on the DBLP database

Algorithm	Accuracy	AUROC
Majority vote	0.697 (0.001)	0.500 (0.000)
RELAGGS	0.732 (0.003)	0.630 (0.013)
N-RELAGGS	**0.739 (0.002)**	**0.668 (0.003)**

The scores are the average of two experiments runs with 500,000 ($\sim$ 69%) randomly selected instances as training set and 224,199 ($\sim$ 31%) instances as test set

The best results are highlighted in bold

Table 11 Accuracy for the different algorithms and databases when classified by a random forest

Algorithm	Trains	Mutagenesis 188	Mutagenesis 42	Carcinogenesis
Aleph	0.90 (0.12)	0.665 (0.009)	0.950 (0.100)	0.553 (0.007)
RSD	0.75 (0.16)	0.767 (0.064)	0.811 (0.088)	0.6088 (0.030)
Treeliker	0.75 (0.27)	0.660 (0.009)	0.833 (0.058)	0.596 (0.019)
Wordification	0.70 (0.24)	0.676 (0.010)	0.833 (0.058)	0.608 (0.056)
RELAGGS	0.75 (0.00)	0.872 (0.030)	0.836 (0.116)	0.620 (0.054)
N-RELLAGS	0.65 (0.20)	0.910 (0.046)	0.856 (0.095)	0.611 (0.033)

The reported values are the average and the standard deviation over the five-fold cross-validation

Table 12 Differences in predictive performance of N-RELAGGS/ Fix N-RELAGGS compared to the other algorithms on the data sets. Each tuple represents better performance, worse performance and equal performance with the numbers in brackets representing the number of significant differences

Data set	N-RELAGGS			Fix N-RELAGGS		
	Better	Worse	Equal	Better	Worse	Equal
Trains	8	6	2	9	5	2
muta 42	13(1)	3	0	11(1)	5	0
muta 188	14(3)	2	0	15(3)	1	0
Carcinogenesis	10(2)	5	1	12(2)	4	0
imdb top	12(5)	3	1	14(5	2	0
movielens	16(10)	0	0	16(10)	0	0

transformation for the RELAGGS and N-RELAGGS algorithms is able to complete the data set. This is primarily due to the potential for parallelization and integrated checkpointing, which enables the algorithm to transform multiple small batches of the database independently on different nodes as well as continuing the computation after termination with a minimal overhead.

Table 10 yields the performance of the experiments on the DBLP data set. The presented results are the average values of the performance scores of two runs with a randomly sampled test set of 224199 instances on the data set. For now, just one set of hyperparameters is used. The classifier has one hidden layer with 100 neurons and λ_{prior} and $\lambda_{\text{posterior}}$ are set to 1.0.

4.5 Feature extraction

As mentioned in Sect. 3.4, we can extract intermediate layers of the N-RELAGGS model and use them to transform relational data instances into single data vectors. This propositionalized representation can then be used as input for other algorithms, for instance, we can compare the performance of the different propositionalizations when classified using a random forest classifier, as shown in Table 11. This experiment is run with the scikit-learn (Pedregosa et al., 2011) implementation of the random forest classifier and all parameters set to the default values.

5 Discussion

The results presented in Sect. 4.3 show the superior predictive performance of our proposed N-RELAGGS algorithm, compared to other state-of-the-art algorithms. The average ranks presented in Table 7 already show that N-RELAGGS is among the best performing algorithms, but with Table 8 we prove that N-RELAGGS performs significantly better than most compared algorithms and that no other algorithm performs significantly better than N-RELAGGS.

Considering the comparisons presented in Fig. 3, we see that the performance regarding the AUROC of N-RELAGGS is significantly better than all other algorithms, except for the regular RELAGGS. Combining the performance differences in Table 12 shows that the number of positive comparisons is at least double the number of negative comparisons for all data sets except trains.

In Sect. 4.4 we introduce the very large DBLP database and try to run all algorithms on it. So far only our implementation was able to complete a run on the database, due to its high parallelizability. The results presented in Table 10 show the superior performance of N-RELAGGS compared to RELAGGS.

Comparing RELAGGS to N-RELAGGS, especially Fix N-RELAGGS, which resembles the structure of RELAGGS more closely, gives no statistically significant difference. Although RELAGGS outperforms N-RELAGGS just on two of six data sets, it ranks on average below N-RELAGGS, which suggests that N-RELAGGS performs better than RELAGGS.

We show that N-RELAGGS is a regular propositionalization algorithm with the experiment in Sect. 4.5. We train the network independent of the actual predictor and select an intermediate layer as propositional representation of the database. This way we can

transform the whole relational database into a propositional representation, i.e. propositionalize the database. Table 11 suggests that the predictive performance of the N-RELAGGS propositionalization is similar to the performance of other algorithms. In terms of the sum of ranks, RELAGGS is slightly superior to N-RELAGGS, however, in the direct comparison, each "win" on two data sets each. No attempts have been made yet to optimize the hyperparameters of N-RELAGSS with respect to the subsequent random forest classifier.

6 Conclusion and Future Work

In this paper we introduce the idea of composite aggregate functions as a way to build a trainable aggregate function from a simple unparameterized aggregate function and two parameterized mappings, and we present a neural network based implementation of them. By combining multiple of those aggregation layers in a network architecture similar to the aggregation scheme of the RELAGGS algorithm, we construct the N-RELAGGS network.

We compare the predictive performance of our proposed N-RELAGGS algorithm with multiple baseline methods, including the structurally similar RELAGGS algorithm, on seven relational data sets and show the advantage of trainable aggregate functions over static aggregate functions. Additionally, we present the DBLP database as a large benchmark data set for propositionalization and relational learning.

A possibility of future improvements is the incorporation of other base aggregate functions into the network architecture. One such possible option is the addition of recurrent neural networks as aggregate functions (Pensel & Kramer, 2020).

Acknowledgements This research has been partially funded by the Federal Ministry of Education and Research of Germany in the framework of Lehren, Organisieren, Beraten: Gelingensbedingungen von Bologna (LOB) (project number 01PL17055). Parts of this research were conducted using the supercomputer Mogon and/or advisory services offered by Johannes Gutenberg University Mainz (hpc.uni-mainz.de), which is a member of the AHRP (Alliance for High Performance Computing in Rhineland Palatinate, www.ahrp.info) and the Gauss Alliance e.V. The authors gratefully acknowledge the computing time granted on the supercomputer Mogon at Johannes Gutenberg University Mainz (hpc.uni-mainz.de).

Funding Open Access funding enabled and organized by Projekt DEAL. This research has been partially funded by the Federal Ministry of Education and Research of Germany in the framework of Lehren, Organisieren, Beraten: Gelingensbedingungen von Bologna (LOB) (project number 01PL17055).

Availability of data and material The DBLP data set is available at https://github.com/kramerlab/n-relaggs/Data/DBLP. The other data sets are publicly available in the relational dataset repository (https://relational.fit.cvut.cz/) and at http://kt.ijs.si/janez_kranjc/ilp_datasets/imdb_top.sql.

Declarations

Conflict of interest Kristian Kersting (TU Darmstadt)

Code availability The source code for the implemented N-RELAGGS algorithm is available at https://github.com/kramerlab/n-relaggs.

References

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., & Devin, M. (2015). Tensorflow: Large-scale machine learning on heterogeneous systems.

Bouckaert, R. R., & Frank, E. (2004). Evaluating the replicability of significance tests for comparing learning algorithms. In *Pacific-Asia conference on knowledge discovery and data mining*, (pp. 3–12). Springer.

De Raedt, L. (2008). *Logical and relational learning*. Springer Science & Business Media.

Debnath, A. K., LopezdeCompadre, R. L., Debnath, G., Shusterman, A. J., & Hansch, C. (1991). Structure-activity relationship of mutagenic aromatic and heteroaromatic nitro compounds. Correlation with molecular orbital energies and hydrophobicity. *Journal of Medicinal Chemistry, 34*(2), 786–797.

Demšar, J. (2006). Statistical comparisons of classifiers over multiple data sets. *The Journal of Machine Learning Research, 7*, 1–30.

Khosravi, H., Schulte, O., Hu, J., & Gao, T. (2012). Learning compact Markov logic networks with decision trees. *Machine Learning, 89*, 257–277.

Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint.* arXiv:1412.6980.

Kramer, S., Lavrač, N., & Flach, P. (2001). Propositionalization approaches to relational data mining. *Relational data mining*, (pp. 262–291).

Krogel, M.-A., & Wrobel, S. (2001). Transformation-based learning using multirelational aggregation. In *Inductive Logic Programming: 11th International Conference, ILP 2001 Strasbourg, France, September 9–11, 2001 Proceedings 11*, (pp. 142–155). Springer.

Kuželka, O., & Železnỳ, F. (2011). Block-wise construction of tree-like relational features with monotone reducibility and redundancy. *Machine Learning, 83*, 163–192.

Lavrač, N., Škrlj, B., & Robnik-Šikonja, M. (2020). Propositionalization and embeddings: Two sides of the same coin. *Machine Learning, 109*, 1465–1507.

Michie, D., Muggleton, S., Page, D., & Srinivasan, A. (1994). To the international computing community: A new east-west challenge. *Distributed email document available from* http://www.doc.ic.ac.uk/shm/Papers/ml-chall.pdf.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., et al. (2011). Scikit-learn: Machine learning in python. *The Journal of Machine Learning Research, 12*, 2825–2830.

Pensel, L., & Kramer, S. (2020). Forecast of study success in the stem disciplines based solely on academic records. In *Machine learning and knowledge discovery in databases: international workshops of ECML PKDD 2019, Würzburg, Germany, September 16–20, 2019, Proceedings, Part I*, (pp. 647–657). Springer.

Perovšek, M., Vavpetič, A., Cestnik, B., & Lavrač, N. (2013). A wordification approach to relational data mining. In *Discovery science: 16th international conference, DS 2013, Singapore, October 6–9, 2013. Proceedings 16*, (pp. 141–154). Springer.

Schouterden, J., Davis, J., & Blockeel, H. (2019). Lazybum: Decision tree learning using lazy propositionalization. In *International conference on inductive logic programming*, (pp. 98–113). Springer.

Šourek, G., Manandhar, S., Železnỳ, F., Schockaert, S., & Kuželka, O. (2017). Learning predictive categories using lifted relational neural networks. In *Inductive logic programming: 26th international conference, ILP 2016, London, UK, September 4–6, 2016, Revised Selected Papers 26*, (pp. 108–119). Springer.

Srinivasan, A. (2001). The aleph manual.

Srinivasan, A., King, R. D., Muggleton, S. H., & Sternberg, M. J. (1997). Carcinogenesis predictions using ilp. In *International conference on inductive logic programming*, (pp. 273–287). Springer.

Tang, J., Zhang, J., Yao, L., Li, J., Zhang, L., & Su, Z. (2008). Arnetminer: extraction and mining of academic social networks. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, (pp. 990–998).

Uwents, W., & Blockeel, H. (2008). Learning aggregate functions with neural networks using a cascade-correlation approach. In *International conference on inductive logic programming*, (pp. 315–329). Springer.

Uwents, W., Monfardini, G., Blockeel, H., Gori, M., & Scarselli, F. (2011). Neural networks for relational learning: An experimental comparison. *Machine Learning, 82*, 315–349.

Wang, Q., Mao, Z., Wang, B., & Guo, L. (2017). Knowledge graph embedding: A survey of approaches and applications. *IEEE Transactions on Knowledge and Data Engineering, 29*(12), 2724–2743.

Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., & Philip, S. Y. (2020). A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems, 32*(1), 4–24.

Železnỳ, F., & Lavrač, N. (2006). Propositionalization-based relational subgroup discovery with rsd. *Machine Learning, 62*, 33–63.