FRI Data Science Project Competition 2024
Final report

One-size-fits-all Relational Database Predictor

Valter Hudovernik, Martin Jurkovič and Uroš Kozole *in collaboration with PC7*

Abstract

In this report we present our approach to relational database prediction using graph neural networks[1]. We design a general framework for tackling predictive tasks on relational data and compare it to a manual feature engineering pipeline created by a data scientist using strong baseline models such as XGBoost. We showcase that our approach is able to beat or match the performance of tabular learners without any feature engineering. Our approach is especially promising when the underlying dataset has a complex relational structure. This leads us to the conclusion that our method promises a one-size-fits-all relational database predictor.

Keywords
Graph Neural Networks, Heterogeneous Graphs, Relational Learning, XGBoost

Advisors: prof. dr. Erik Štrumbelj

Introduction

Relational databases are the basis of data storage, holding a vast amount of the world's information [1]. This includes customer data, financial records, scientific observations and social media interactions. However, extracting knowledge and future trends from these databases is often complex. Existing methods typically require significant expertise in the specific data domain and manual analysis of the database schema. This creates a barrier for those who want to quickly explore and utilize their data. While tabular learning excels at analyzing single tables, it struggles with the interconnected nature of relational databases.

A typical approach to relational machine learning includes a manual feature engineering step, where a data scientist uses domain knowledge to join and or aggregate tables to generate features in single table format suitable for tabular learning. This approach suffers from several drawbacks, including slow feature engineering that demands significant human effort. The choices of features are subjective and might not capture the most relevant information. Additionally, manually exploring all possible features is infeasible, while aggregating data into a single table results in a loss of information. Furthermore, the final solution does not generalize well to other problems involving different databases. That is why we were motivated by the company PC7 d.o.o. to research the one-size-fits-all relational database predictor[1]. The idea is to train a model that could fit any relational database structure and adapt to any ML task.

Classical feature extraction and engineering approaches have been gradually replaced by deep learning in fields like computer vision [2] and natural language processing [3]. Similarly, a framework for relational deep learning on graphs has been recently proposed by [4]. Due to the challenges of traditional relational ML approaches, we turn to graph neural networks and relational deep learning as a viable option.

Related Work

Graph neural networks (GNNs) are neural network architectures that operate on graph-structured data. A graph is a structure that represents the relations between a collection of entities. Enities are usually represented with nodes (vertices) and relations with edges (links). Edges that connect the nodes can be directed or undirected. Both nodes and edges can contain attributes. There are 3 general types of prediction tasks on graphs: graph-level, node-level and edge-level. In a graph-level task, we predict a single property for a whole graph. Similarly for node-level and edge-level tasks we predict some property for each node or each edge respectively. All of the following problems can be solved with a single model class, the GNN.[5]

A GNN is an optimizable transformation on all attributes of the graph (nodes, edges and global-context) that preserves graph symmetries (permutation invariances). GNNs adopt a "graph-in, graph-out" architecture, meaning that these model

[1]Our work is available as a github repository (https://github.com/uroskozole/PC7-DB_prediction).

types accept a graph as input, with information loaded into its nodes, edges and global-context, and iteratively transform these into latent embeddings, which contain information about the attributes as well as the structure of the graph.

The main idea in construction of GNNs is propagation of information between nodes via edges. During inference on the input graph, nodes receive information from their neighboring nodes in several iterations to reach their target embedding. Each iteration consists of: 1) nodes collecting the embeddings of their neighbors (message passing step), 2) nodes aggregating the embeddings received from their neighbors (aggregation step) and 3) combining their current embedding with the result of the aggregation step and updating their embedding (update step) [5]. Neural networks are incorporated at the aggregation step (they transform the incoming embeddings before the aggregation) and for combining the current embedding and the aggregation result. After the last iteration, the graph with resulting embeddings is returned as the output graph. In essence, the neural network part of the GNNs is tasked with learning how to calculate the target embedding of each node based on the node's neighborhood.

Decisions we make when choosing how to incorporate the neural networks yield different architectures. Below we review the architectures we use in this project.

GraphSAGE [6] is a general inductive learning GNN framework that stands for sampling and aggregation of node representations. It is arguably the most impactful GNN arhitecture that generalizes the transductive spectral convolution on graphs to message passing using learnable aggregation functions. This allows the model to generalize to previously unseen graph structures, which the previous methods did not allow. They define *message-passing* - a parameterized composition of functions defining how the information is propagated throughout the graph based on its structure.

Graph Attention Networks (GAT) [7] are GNN architectures which use masked self-attention layers as the message passing mechanism. The attention mechanism represents a learnable permutation invariant function. It can also be applied to graph nodes having different degrees by allowing the network to learn to attend to the node's most important neighbours. This makes the GAT model directly applicable to inductive learning problems, allowing the model to generalize to unseen graphs.

EdgeCNN [8], was designed to solve the task of learning features on point clouds, scattered collections of points. Point clouds are commonly used in 3D modelling by LiDAR scanners and stereo reconstruction. EdgeCNN constructs a local neighborhood graph and apply convolution-like operations on the edges connecting neighboring pairs of points. Unlike graph CNNs, the computational graph is not fixed but is dynamically updated after each layer of the network. With dynamic updates the receptive field of the model is as large as the diameter of the point cloud, while remaining sparse. Their architecture learns how to dynamically construct the computational graph, as opposed to other popular GNN methods.

Graph Isomorphism Network (GIN) [9] is a simple GNN architecture for which the authors show that its representation learning power is superior to other popular architectures. They arrive at the conclusion that some graph structures cannot be distinguished by popular GNN variants, such as GraphSAGE. The key novelty behind this approach is that it enables the model to differentiate graphs that are not isomorphic to each other, a difficult problem from discrete mathematics.

Methods

Dataset Graph Representation

To define our relational learning task we must convert the relational databases from their metadata schema to the corresponding graph representation. To account for different structure and types of individual tables, we choose the heterogeneous graph representation. 'Heterogeneous graphs' are graphs with different types of nodes and edges. Our algorithm for converting a relational database to a heterogeneous graph is outlined in Algorithm 1.

Algorithm 1 Building a Graph from a Relational Database

```
Require: Relational database $D = \{T_1, ...T_n\}$
Require: Relational schema $S\{R_1, ...R_m\}$
 1: Initialize Heterogeneous graph $G$
 2: for $T_i$ in $D$ do
 3:     for row in $T_i$ do
 4:         Add row to $G$ as node $n$
 5:         $n.type \leftarrow type(T_1)$
 6:         $n.features \leftarrow row.columns$
 7:     end for
 8: end for
 9: for $R_i$ in $S$ do
10:     $(fk_1, fk_2) \leftarrow R_i$
11:     Add edges $e = (R_i.T_1[fk_1], R_i.T_2[fk_2])$ to $G$
12: end for
```

Datasets

We select 3 relational datasets: *Rossmann Store Sales*, *Biodegradability* and *PKDD'99 Financial dataset*. We report the summary of the datasets in Table 1.

The *Rossmann store sales* dataset [10] features historical sales data for 1,115 Rossmann stores. Each store has a set of both numerical and categorical features. The dataset consist of two tables connected by a single foreign key. This makes it the simplest type of relational dataset. The first table contains general information about the stores and the second contains sales-related data. The standard task is to predict the number of customers on a specific day for a given store.

The *Biodegradability* dataset [11] comprises a collection of chemical structures, specifically 328 compounds, each labeled with its half-life for aerobic aqueous biodegradation. This dataset is intended for regression analysis, aiming to predict the biodegradation half-lives activity based on the chemical features of the compounds.

Dataset Name	# Tables	# Rows	# Columns	# Relation types	Hierarchy Type
Rossmann Store Sales	2	59.085	16	1	Linear
Biodegradability	5	21.895	6	5	Multi Child & Parent
Financial	8	1.079.680	55	8	Multi Child & Parent

Table 1. A summary of the 3 evaluation datasets. The number of columns represents the number of non-id columns.

PKDD'99 Financial dataset [12] contains 606 successful and 76 unsuccessful loans along with their information and transactions. The standard task is to categorize the loan outcome at the time of the loan start, as either successful or unsuccessful.

Heterogenous GNNs

Our problem inherently requires dealing with entities of different types originating from different tables of the relational database. As outlined previously this results in 'heterogeneous graphs', that require so called 'Heterogeneous GNNs' (H-GNNs). In short, H-GNNs deal with different node and edge types by separating the message-passing parameters for each type of relation, where a relation is defined with a tuple of (start node, edge type, end node). Naturally this allows more expressivity, but increases parameter count and complexity.

We convert the commonly used GNN architectures *GraphSAGE*, *GIN*, *EdgeCNN*, *GIN* from their homogeneous to heterogeneous variants and add a multi-layer perceptron (MLP) to the final layer representation, as seen in Figure 1.

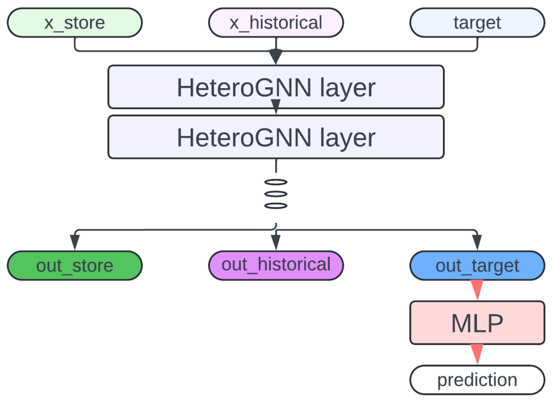

Figure 1. Example of complete HeteroGNN architecture as used on the Rossmann Store Sales dataset. We add an additional target node to the computational graph, and connect it to the corresponding nodes. The model retains separate representations for each node type, which communicate based on the underlying GNN layer. For a detailed visualization of one such layer refer to Figure 2.

The H-GNN model consists of a stack of heterogeneous layers, which define the message passing functions based on the underlying GNN arhitecture. An example of a heterogeneous layer's architecture for the *Rossmann* dataset is shown in Figure 2.

Baseline

To evaluate the performance of GNNs on relational data we have to compare the performance with some baseline. As a baseline we simulate the approach of a data scientist by handcrafting a machine learning pipeline, where we transform the data from a relational structure into a single table. We perform manual feature engineering for each dataset and use linear and XGBoost models as baselines. We note that XGBoost is a state-of-the-art tabular learner [13] with a custom-made pipeline for each dataset, while we train all GNN models with a single general framework, to simulates a real-world use case of one-size-fits-all relational models.

Results

We evaluate the models on 2 regression and one classification task on the 3 relational datasets. We use five H-GNN models trained on the graphs abtained from the relational datasets and a linear regression and XGBoost model as a baseline trained on manually constructed feature engineering pipelines. We report the results of the experiments in Table 2. We report root mean squared error (RMSE) for the regression tasks and F1 score for the classification task.

Method	Rossmann	Biodegradability	Financial
H-GAT	130.75 ± 0.08	$\mathbf{0.96 \pm 0.02}$	0.40 | 0.67*
H-GATv2	135.11 ± 0.07	1.12 ± 0.02	**0.67** | **0.80***
H-GIN	132.44 ± 0.06	1.13 ± 0.01	0.14 | 0.31*
H-GraphSAGE	129.47 ± 0.07	1.08 ± 0.02	0.29 | 0.50*
H-EdgeCNN	126.74 ± 0.07	1.07 ± 0.01	0.25 | 0.40*
XGBoost	$\mathbf{124.83 \pm 0.01}$	1.39 ± 0.06	0.40 | 0.67*
Linear Model	290.39 ± 0.01	1.35 ± 0.02	0.40 | 0.33*

Table 2. GNN (top) vs. baseline (bottom) RMSE for Rossmann and Biodegradability datasets and F1-score for Financial dataset. For financial data we report two F1-scores, where * denotes the use of target minority class oversampling.

All GNN models outperform XGBoost on the *Biodegradability* and the best models outperform and the rest match the performance of the baseline models on the *Financial* dataset. On the simpler *Rossmann* dataset, the models perform competitively to the strong XGBoost baseline.

We note that the *Rossmann* dataset consists of only two tables that can be joined into one table without loss of information. This makes it effectively a tabular dataset, giving an edge to the XGBoost model, optimized for tabular learning. Still the GNN models, without any additional feature processing, were able to almost match the XGBoost model.

For the *Biodegradability* dataset all of the GNN models were able to outperform the baseline models. We believe this

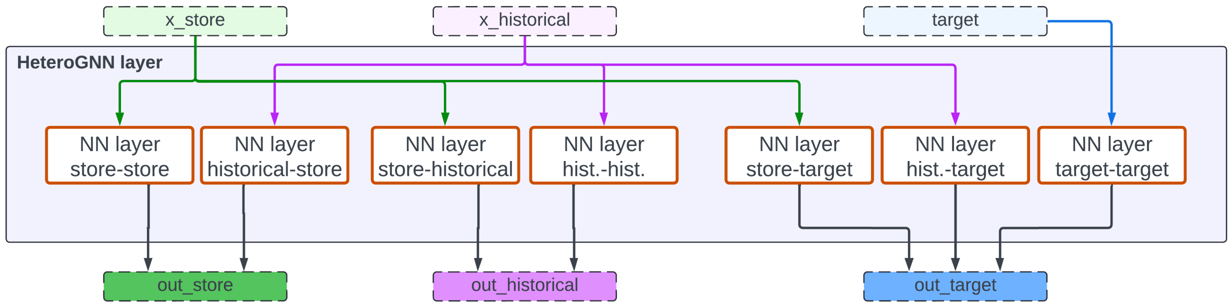

Figure 2. Example of one layer of heterogenous GNN – Rossmann Store Sales dataset. Each type of relation is fully defined by start-end node combination. For NN layers we tested layers of GNNs described in the Methods section.

is because the dataset consists of molecule data, which is best described with graphs making GNNs better suited to exploit this structure than a tabular baseline model.

For the *Financial* dataset we report the F1 score as the target class is unbalanced. We report two scores for each model, one with and one without oversampling the minority class. We see that the best GNN outperform or match the performance of the XGBoost model.

Conclusion

We introduce an approach to relational database prediction using graph neural networks (GNNs), effectively addressing relational predictive tasks without the need for manual feature engineering. By converting relational databases to heterogeneous graph representations, our GNN models capture complex relationships within the data and outperform or match traditional methods like XGBoost on three diverse datasets.

While our method does not require manual feature engineering, this comes at a cost of extensive upfront computation while converting the relational database to its graph representation. Furthermore, the training time of deep models is orders of magnitude longer than that of classical approaches. This also limits our ability to perform extensive hyper-parameter optimisation, which is a necessary step in order to further optimize the performance of the models. However, despite these limitations our results demonstrate that our GNN-based framework is a promising one-size-fits-all solution for relational database prediction, simplifying the prediction task from the user's perspective.

References

[1] DB-Engines. DBMS popularity broken down by database model, 2023.

[2] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 2015.

[3] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[4] Matthias Fey, Weihua Hu, Kexin Huang, Jan Eric Lenssen, Rishabh Ranjan, Joshua Robinson, Rex Ying, Jiaxuan You, and Jure Leskovec. Relational deep learning: Graph representation learning on relational databases. *arXiv preprint arXiv:2312.04615*, 2023.

[5] Benjamin Sanchez-Lengeling, Emily Reif, Adam Pearce, and Alexander B. Wiltschko. A gentle introduction to graph neural networks. *Distill*, 2021.

[6] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. *Advances in neural information processing systems*, 2017.

[7] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph Attention Networks. *International Conference on Learning Representations*, 2018.

[8] Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E Sarma, Michael M Bronstein, and Justin M Solomon. Dynamic graph cnn for learning on point clouds. *ACM Transactions on Graphics (tog)*, 2019.

[9] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *International Conference on Learning Representations*, 2019.

[10] Will Cukierski. Rossmann store sales, 2015.

[11] Hendrik Blockeel, Sašo Džeroski, Boris Kompare, Stefan Kramer, Bernhard Pfahringer, and Wim Laer. Experiments in predicting biodegradability. *Applied Artificial Intelligence*, 1999.

[12] Petr Berka. Pkdd'99 financial dataset, 2001.

[13] Léo Grinsztajn, Edouard Oyallon, and Gaël Varoquaux. Why do tree-based models still outperform deep learning on typical tabular data? *Advances in neural information processing systems*, 35:507–520, 2022.