FRI Data Science Project Competition 2024
Interim report

One-size-fits-all Relational Database Predictor

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

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:

1. Slow feature engineering which requires significant human effort
2. Feature choices are subjective and may not capture the most relevant information
3. Manually exploring all possible features is infeasible
4. Loss of information by aggregation of data into a single table

That is why we were motivated by the company PC7 d.o.o. to research the one-size-fits-all relational database predictor. 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.

Relevant tools

PyTorch Geometric

PyTorch Geometric (PyG) [5] is a python library built on top of PyTorch. PyG simplifies the development and training of Graph Neural Networks (GNNs) for various applications involving structured data analysis. It provides a comprehensive suite of tools specifically designed for GNNs, making it an ideal choice for our research.

Crucially, PyG offers implementations of diverse GNN architectures. This allows us to explore a wide range of approaches for the task at hand. Additionally, the library features optimized message passing mechanisms, a core concept in GNNs. This ensures efficient training and inference when working with graph-structured data. Furthermore, PyG's support for scalable data handling enables us to work with large and complex datasets.

DeepSNAP

DeepSNAP is a Python library designed to streamline Heterogeneous Graph Neural Network (GNN) development and application across various domains. DeepSNAP integrates graph manipulation libraries like NetworkX with deep learning frameworks like PyTorch Geometric. It also solves the following challenges when training GNNs.

One challenge is the need for complex graph manipulations during GNN training. These manipulations may include feature computation, pretraining, and subgraph extraction, sometimes required on a per-iteration basis. DeepSNAP provides robust tools to handle these tasks efficiently.

Another challenge is the lack of standardized pipelines in existing frameworks. Implementing standard pipelines for node, edge, link, and graph-level tasks (under inductive or transductive settings) is often left to the user. DeepSNAP offers pre-defined pipelines, reducing repetitive coding and facilitating fair comparisons between different GNN models. These pipelines may also include functionalities like data splitting for link prediction tasks.

Furthermore, real-world graphs are frequently heterogeneous, containing diverse node and edge types. DeepSNAP offers comprehensive support for handling such heterogeneous graphs, including efficient data storage and flexible message passing mechanisms, which might be lacking in other frameworks.

RelBench

RelBench [4] is a python package for benchmarking and evaluation in the context of Relational Deep Learning. It covers everything from data loading, predictive task specification, (usually temporal) data splitting, data-to-graph transformation, GNN predictive model development, as well as evaluation via a standardised evaluation protocol.

Model module uses PyG while both Data and Evaluation module are framework-agnostic thus allowing broad compatibility.

Typical RelBench workflow consists of (1) loading the database and processing it into the standardised graph format, (2) specifying the task and loading relevant tables to form the train, test and validation tables, (3) training the model, which can be done with RelBench's Model module or elsewhere, and (4) evaluation of the trained module with the Evaluation module.

RelBench beta release includes two databases with two predictive tasks each. **Stack Exchange database** includes tables such as posts, users, votes etc. The two predictive tasks are prediction of user contributions (posts, answers, votes ...) and prediction of question's popularity. **Amazon book catalog database** includes three tables: users, products and reviews. Predictive tasks are prediction of lifetime value of a user and predicting user churn.

Methods & Project Plan

Dataset Selection and Graph Representation

For dataset selection we will use a python package called the Synthetic Data Vault, which primarily focuses on tabular synthetic data generation. They have collected 48 relational datasets with provided metadata schema about the structure and column information. These datasets differ by relational structure, number of rows and column data types in corresponding tables.

We will then convert these databases from their metadata schema to the *pytorch geometric* graph schema. From there we have a collection of ready to train datasets in the graph neural network format. The algorithm for converting a relational database to heterogeneous graph is outlined in Algorithm 1.

Algorithm 1 Building a Heterogeneous Graph from 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 edge $e = (fk_1, fk_2)$ to $G$
12: end for
```

Additionally, we will also use datasets provided in the RelBench evaluation suite.

Predictive Tasks

We will use the pytorch geometric and deepsnap libraries to fit different GNN algorithms on the collected datasets. For each dataset we will specify one or more predictive tasks proposed in [4]. By doing this we will get a good overview of the predictive capabilities of different GNN arhitectures.

Additionally, each of the authors will create a manual feature engineering and predictive pipeline that will not use graph neural networks. This will serve as a baseline score for the predictive tasks.

References

[1] DB-Engines. DBMS popularity broken down by database model, 2023. Available: https://db-engines.com/en/ranking_categories, 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*, 115:211–252, 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] Matthias Fey and Jan Eric Lenssen. Fast graph representation learning with pytorch geometric. *arXiv preprint arXiv:1903.02428*, 2019.