Applied Deep Learning Report

Name: Dominik Pichler

Matriculation Number: 01530718

Course: Applied Deep Learning

Date: 20.01.2025

1. Introduction

In this project I have investigated **Graph Structure-based Fraud Detection** in Financial Transaction Networks with the help of **Geometry** and **Graph Neural Networks** to fight Tax Fraud. Initially inspired by the big issue of *Value-Added-Tax Fraud*, where in 2021 alone approx. 15 billion euros have been stolen by criminals applying Value-Added-Tax Fraud-techniques in the EU according to *Ott (2024)*. Due to lack of publicly available data, I had to switch to something similar: Money Laundry Patterns in Transaction Networks. As I chose a geometry based approach in combination with GNNs, it might still be useful for VAT Fraud as well.

Therefore, I tried to **bring my own method** to detect money laundering in the geometrical structures present in the IBM Transactions for Anti Money Laundering (AML) while orienting myself on the following three papers:

- The geometry of suspicious money laundering activities in financial networks
- Provably Powerful Graph Neural Networks for Directed Multigraph
- Smurf-based Anti-money Laundering in Time Evolving Transction Networks

For this purpose, I am using the following dataset: IBM - Syntetic Transaction Data for Anti-Money-Laundry Dataset: This dataset contains 180 Mio. transactions with a high and low Laundering-Transaction Share that includes the following columns:

Column	Description
Timestamp	The date and time when the transaction occurred.
From Bank	The identifier of the bank from which the funds were sent.
Account	The account number from which the funds were sent.
To Bank	The identifier of the bank to which the funds were sent.
Account	The account number to which the funds were sent.
Amount Received	The total amount of money received in the transaction.
Receiving Currency	The currency in which the amount was received.
Amount Paid	The total amount of money paid in the transaction.
Payment Currency	The currency in which the payment was made.
Payment Format	The method or format used for the payment (e.g., Reinvestment, Cheque, ACH).
Is Laundering	Indicates whether the transaction is suspected of money laundering (0 = No).

2. Implementation / Solution

The core in this project is a neuro-symbolic DL approach, combining symbolic AI (logics / Rule-based approaches) and graph neural networks. My own method consists of combining geometry-based preprocessing with powerful Graph Neural Networks for directed Multi graphs.

2.1. Setup / Data Preparation

I used a dockerized Neo4j database to store the previously mentioned dataset as property graph, for further analysis.

2.2. Geometry-based Preprocessing

For this part, I utilized geometric models to preselect the networks on which the Graph (Neural) Nets are trained and tested.

I expect this to increase the efficiency and performance in correctly classifying Money Laundry Transactions (measured by the *F1 Score*) , as I hoped to increase the quality of the GNNs Training data by filtering out obviously irrelevant sub-graphs). In addition, I expect this to yield an overall lower computational complexity of the GNN Training. In order to do so, I've borrowed some thoughts from Granados et al. (2022)

In order to identify (sub)graphs worth preselecting, I stuck to the general definition of *Money Laundering* by the **Financial Action Task Force (FATF)** that defines it as *the process by which money generated through criminal activity appears to have come from a legitimate source.* So based on the definition above, some characteristic behaviors of money laundry can, and have been derived by the **FATF** and **UNODC**:

- **Rule 1.** The money launderer agent's interactions do not grow rapidly because it is preferable to maintain only few interactions with other agents and to keep them as anonymous or covert as possible.
- **Rule 2.** The money launderer agent does not impose restrictions on the geographical distance required for those interactions. This includes the transference of money between a non-haven jurisdiction and a tax haven jurisdiction.
- **Rule 3.** The money launderer agent does not care how many transactions are needed to clean the illegal money (cycle), resorting to deposits triangulation between the same agents.
- **Rule 4.** The money launderer agent employs the breaking up of large amounts of money into smaller amounts to avoid suspicions. The money is then deposited into one or more bank accounts or other financial instruments either by different persons or by a single person over a period of time.
- **Rule 5.** After a process of money laundering interactions is initiated, it is to be expected that at some point (at least part of) the involved money returns to the money launderer agent completing a cycle.

3. Graph Neural Networks

For comprehensive Analysis, I have implemented and tested multiple GNNs. More specifically, the following:

- GATe (Graph Attention Network with edge features) / GAT
- GINe (Graph Isomorphism Network with edge features)
- PNA (Principal Neighbourhood Aggregation)
- RGCN (Relational Graph Convolutional Network)

The selection has been explicitly designed to incorporate the evolution of GNN Paradigms over time.

GATe (Graph Attention Network with edge features)

GATe is an extension of the original Graph Attention Network (GAT) that incorporates edge features into the attention mechanism. Key characteristics include:

- Attention mechanism that considers both node and edge features together
- Delivers improved performance on graphs with rich edge information
- Possesses the ability to capture complex relationships between nodes and edges

GATe has shown promising results in tasks where edge attributes are crucial, such as molecular property prediction and social network analysis.

GINe (Graph Isomorphism Network with edge features)

GINe is a variant of the Graph Isomorphism Network (GIN) that also takes edge features into account. Notable aspects include:

- Shines at preserving the expressive power of GIN for node and graph-level tasks
- Incorporates edge features in the message passing step
- Suitable for graphs where edge attributes carry important information

This model has been particularly effective in chemical and biological applications where bond types and other edge properties are significant.

PNA (Principal Neighbourhood Aggregation)

PNA is a GNN architecture designed to be more adaptable to **diverse graph structures**. Key features include:

- Uses multiple aggregators (mean, max, min, standard deviation)
- Employs degree-scalers to adjust for varying node degrees
- Combines different aggregators and scalers for robust feature extraction

PNA has demonstrated state-of-the-art performance on various benchmarks, especially in tasks involving **graphs with heterogeneous structures**.

RGCN (Relational Graph Convolutional Network)

RGCN is an extension of Graph Convolutional Networks (GCN) designed to handle multi-relational graphs. Important characteristics include:

- Supports multiple types of relationships between nodes
- Uses relation-specific weight matrices for different edge types
- Effective for knowledge graphs and other multi-relational data

RGCN has been successfully applied in various domains, including **knowledge base completion, entity classification, and link prediction in heterogeneous networks**.

4. Putting everything together & evaluate:

In order to evaluate weather the GBPre is actually beneficial, I chose to use the *F1* Score. While the absolut *F1 Score* is interesting, I want to investigate the relative difference between the models with and without GBPre. I target to achieve higher *F1* Scores with the GBPre models. Nonetheless, an *F1 Score* below 0.5 would be bad, as it would indicate a worse performance than flipping a coin for choosing weather a transaction is money laundering or not.
In order to be able to compare the different models with and without the GBPre I ran every model with and without *Geometry based Preprocessing* All important metrics have been stored and logged via **wand.ai**.

Hyperparameter Tuning

For the Hyperparameter Tuning, I tried the following different sets of Hyperparameter Configurations:

Batch Size	Number of Epochs	Number of Neighbors
4096	50	[50, 50]
4096	50	[100, 100]
4096	50	[150, 150]
4096	100	[50, 50]
4096	100	[100, 100]
4096	100	[150, 150]
4096	150	[50, 50]
4096	150	[100, 100]
4096	**200**	**[150, 150]** *(best one)*
8192	50	[50, 50]
8192	50	[100, 100]
8192	50	[150, 150]
8192	100	[100, 100]

Results

With the best performing Hyperparameter, the models yielded the following performance:

Name	model	n_gnn_layers	best_test_f1	f1/test	f1/train	f1/validation
GBPre_PNA	pna	2	0.418274112	0.334883721	0.936030618	0.100558659
GBPre_RCGN	rgcn	2	0.493943472	0.50831793	0.950727884	0.24009324
GBPre_GAT	gat	2	0.475916607	0.467862481	0.786960514	0.258515284
GBPre_GIN	gin	2	0.242557883	0.397350993	0.853795211	0.131445905
no_GBPre_PNA	pna	2	0.454124904	0.454192547	0.469369485	0.444698703
no_GBPre_RCGN	rgcn	2	0.451086957	0.430457746	0.486665782	0.432363014
no_GBPre_GAT	gat	2	0.454371898	0.455135548	0.467575487	0.450810403
no_GBPre_GIN	gin	2	0.455309396	0.419461502	0.498770156	0.427923844

Conclusion:

Out of all models, **GBPre_RCGN** (RCGN with Geometry based Preprocessing) performed the best. Also, in all cases except for **GBPre_GIN** the models with GBPre outperformed the standard models in the *F1_Test*. This highlights my initial assumption that geometry based preprocessing might acutally be a good enhancement for the deep learning approach (which proved very useful for this kind of problem).

To answer the question of wether Deep Learning was an apppropriate tool for this kind of problem, I would conclude that it definitely is a very good, if not the best approach to my knowledge, as the complexity of those networks is way to big to be processed and analysed otherwise.

Regarding the time-estimates, I was pretty accurate, except for the preprocessing which tool longer than expected due to my underestimation of the complexity. What I also underestimated was the amout of RAM needed to preprocess the data in Neo4j.

Learnings

If I would start over again, I would probably think even more abouth the project structure / the code architecture and begin with the acutal application in mind. Also, I would adhere more strictly to a ML Processframework like *CRISP ML(Q)* for example.

Appendix (A): Efforts so far

A.1 Dataset Collection and Preprocessing

- **Research and Identify Suitable Datasets**: 9 hours
- **Data Cleaning and Preprocessing**: 15 hours
- **Feature Engineering**: 50 hours

A.2 Designing and Building the Graph Neural Network

- **Literature Review on Graph Neural Networks**: 10 hours
- **Network Architecture Design**: 15 hours
- **Implementation of the Network**: 35 hours

A.3 Training and Fine-Tuning the Network

- **Initial Model Training**: 10 hours
- **Hyperparameter Tuning**: 10 hours
- **Validation and Testing**: 15 hours

A.4 Building an Application to Present Results

- **Design User Interface**: 15 hours
- **Develop Application Backend**: 15 hours
- **Integrate Model with Application**: 10 hours

A.5 Writing the Final Report

- **Drafting the Report Structure**: 3 hours
- **Writing and Editing Content**: 4 hours
- **Creating Visualizations and Appendices**: 2 hours

A.6 Preparing the Presentation

- **Design Presentation Slides**: 5 hours
- **Rehearse Presentation Delivery**: 2 hours