Differentiable Inductive Logic Programming for Fraud Detection

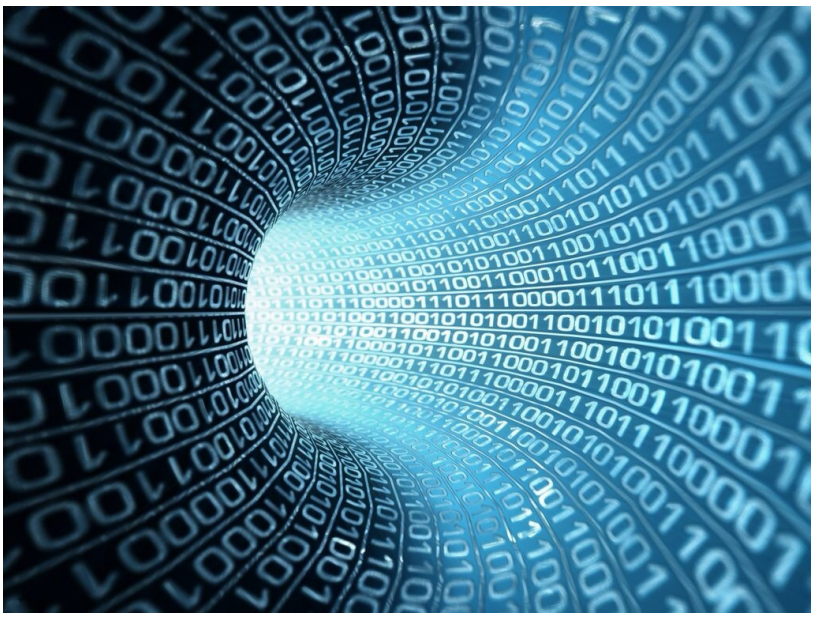
Extracting Explainable Rules for Different Fraud Scenarios

submitted in partial fulfillment for the degree of master of science

Boris Wolfson
13551884

Master Information Studies
data science
Faculty of Science
University of Amsterdam

Submitted on 30.06.2024



	UvA Supervisor
Title, Name	Dr. Erman Acar, ILLC
Affiliation	UvA Supervisor
Email	e.acar@uva.nl

ABSTRACT

Current trends in Machine Learning prefer explainability even when it comes at the cost of performance [17]. Explainable methods are particularly important in the field of Fraud Detection. This work examines the applicability of Differentiable Inductive Logic Programming (∂ILP) as an explainable AI approach for Fraud Detection. We find that ∂ILP is not scalable and requires data manipulation to adjust the tabular data to the expected format of background facts statements. We proposed how to convert numerical data to be used by ∂ILP. While we could replicate the result classification techniques such as Decision Tree, and Deep Symbolic Classification on the PaySim dataset, ∂ILP did not show any benefit as opposed to these methods. Still, we believe a promising research direction exists, where methods similar to ∂ILP can outperform the classical methods for Fraud Detection, as we showed in the scenarios of relationships between different fraudulent agents or inventing a rule requiring learning recursive structures.

KEYWORDS

Rule Learning, Fraud Detection, Neuro-Symbolic AI, Explainable AI

GITHUB REPOSITORY

https://github.com/wolfbrr/ThesisDS

1 INTRODUCTION

It is said that the money fraud problem is as old as money itself [9]. Fraud contributes not only to financial loss but also impacts people, industries, entities, services, and the environment. According to the Annual Fraud Report from UK Finance, the overall fraud losses in the UK in 2022 added up to $ 1.2 billion, of which fraud losses across payment cards and remote banking had a share of $ 726.9 million [11].

With the shift towards online marketing as a result of the Covid-19 pandemic the amount of cybercrime increased significantly [15]. One way of tackling the problem of fraud is by detecting and preventing fraudulent online transactions. Fraud has always been a part of every society and it has been evolving throughout the decades. Criminals continued to develop new sophisticated methods making their detection challenging.

Some fraudulent transactions involve multiple actors, companies continuously invest in sophisticated protection tools for recognizing fraud activity patterns[1] involving different players. Money laundering is another example of fraudulent activity distributed between multiple agents. The work of Granados and Vargas [12] approaches the money laundering problem as a geometric graph analysis, where one should recognize cycles in money transactions.

Fraudulent action detection has been addressed with knowledge-driven approaches, data-driven approaches, and combinations of these two. Knowledge-driven approaches implement rules assisting in the detection of malicious activity and provide the explainability of the detection made.

[1] https://www.radial.com/insights/understanding-triangulation-fraud

In fraud detection, it is important to discuss the explainability of AI methods. In general, the explainability of the AI system is an important factor in modern society. Understanding the AI system's decision plays an important role in preventing racial biases and incorporating fairness. Particularly in the fraud-related field, a lack of fairness can be exploited by fraudsters [23]. On the other hand, data-driven approaches such as neural networks, bring autonomy and good qualitative performance but lack the explainability component. The goal, therefore, is to combine the strength model a concept that recognizes fraud and can explain the result at the same time [14].

Neurosymbolic (NeSy) approaches provide a combination of the strength of data and knowledge, bringing explainability and qualitative performance. In this field, much of the work done is related to simple rule extraction in the $IF - THEN$ form. This requires a lot of data to train the model, and it is still limited in generalization. Generalization is important in the fraud detection area since the available data is usually highly unbalanced towards non-fraudulent transactions.

One approach that deals well with small data sets and is known for its ability for generalization is the Inductive Logic Programming (ILP) paradigm. ILP provides a set of rules that explains the input data set. The data set consists of positive and negative examples, and the program entails all the positive examples and does not entail any negative ones. Thus, from the machine learning point of view, the ILP system is ultimately a binary classifier that not only explains the data itself but is robust enough to generalize on unseen data. Having said that, the main disadvantage of ILP is that it does not deal well with noisy, erroneous, or ambiguous data [10].

Recently, a few works emerged suggesting a neural symbolic extension for implementing the ILP, one of which was a Differentiable Inductive Logic Programming ∂ILP [10]. Evans and Grefenstette [10]$showedthat$∂ILP can provide generalization when applied to noisy data. This thesis will focus on ∂ILP implementation for fraud detection, which to our best knowledge was not yet implemented in this area. Therefore our central research objective in this thesis is:

RQ1 To what extent a neurosymbolic ILP method can provide explanations in the domain of fraud detection?

To answer this question, the following sub-questions are answered:

RQ2 What is the level of performance concerning other traditional approaches, such as Decision Trees, and Deep Symbolic Classification?

RQ3 What is the trade-off between the size of the rule vs its performance?

RQ4 To what extent ∂ILP can provide explanatory rules by using Recursive Structures for detecting relationships between different agents?

To address these questions, first, we developed our synthetic data set to test the performance of ∂ILP on a controllable dataset, and then we trained and evaluated the model on the PaySim dataset[18]. The results are compared against results achieved with existing

explainable methods such as Decision Trees and Symbolic Classification. Finally, we test ∂ILP on more complex scenarios such as *Fraud Relationhsip* and *Chain of Fraud* for testing its ability to create Recursive Structures.

2 RELATED WORK

Multiple approaches to derive rules from the tabular data exist - the most famous one is the Decision Tree (DT) approach, where the $IF - THEN - ELSE$ set of rules is applied to split the data into different categories [4]. Although DTs provide explanatory rules, these rules come in multiple spits, affecting interpretability. In addition, as was shown by Bengio et al. [2] DTs are known to overfit and lack generalization.

Recent approaches use Neural Networks to derive the set of rules, by implementing AND, and OR logical operators by two consecutive layers respectively. The Decision Rules Network (DR- Net), and the Relational Rule Network (R2N) for example, provide a set of rules in a disjunctive normal form (DNF) as shown in Equation 1 [16, 24].

$$IF \text{ A or B or C ... } THEN \text{ ...} \tag{1}$$

The work of Collery et al. [7] extended this approach by applying an R2N layer as a convolutional window, for discovering patterns for the classification of sequential data. These approaches deliver an inherently explainable set of rules but cannot derive recursive predicates or invent a new predicate. The size of the derived rule formula expresses the importance of recursion. In addition, recursion is useful in generalizing from small datasets. The ability of predicate invention allows algorithms to learn patterns without explicit input from domain experts.

2.1 Inductive Logic Programming

ILP is a Machine Learning approach that learns first-order rules from relational structured data. The data consists of a set of facts and positive and negative instances of a target predicate. The learned rules are a so-called logic program that defines a target predicate, which is true for all the positive and false for the negative examples [20].

ILP lacks the strength of a deep neural network to map a complex problem and is known to be weak in dealing with noisy data. A few neurosymbolic extensions to the ILP were compared and discussed in [3]. The basic idea behind the extensions is to generate a set of initial rules and then to learn the relevant ones. One is ∂ILP, presented in this thesis. The other is differentiable Neural Logic ILP (dNL-ILP) [21].

Both dNL-ILP and ∂ILP learn target predicates based on the facts defined by the set of predicates and auxiliary predicates. Domain experts usually define auxiliary predicates. The difference between the two approaches lies in how the initial set of rules is generated. ∂ILP has several restrictions in so-called language bias, the template that defines how to generate the rules. dNL-ILP has fewer restrictions and is thus supposed to be more scalable. A third approach is Meta_{Abd} [8]. Meta_{Abd} implements background knowledge as a set of rules, instead of facts and works on images, instead of positive and negative examples. All three are capable of implementing recursion and inventing predicates. In this work, we opted to start exploring ∂ILP because it has good documentation and established research.

2.2 Fraud Detection

Fraud detection is a binary classification problem of anomaly/-suspicious activity detection. Therefore, multiple classification approaches are available to tackle this problem. For instance, basic logistic regression or DTs can be used and are considered to be explainable methods. More complex and accurate LSTM and the XGBoost are examples of non-explainable AI approaches [2].

Hajek et al. [13] compares the performance of the XGBoost framework to other conventional methods, applied on the PaySim dataset, and recent work by Visbeek et al. [27], introduced a Deep Symbolic Classification (DSC) framework, and applied it to fraud detection problems. The DSC learns a set of rules, by extending Deep Symbolic Regregions (DSR)[22], and results are summarised in Tabel 1.

Table 1: DSC and XGBoost performance on PaySim dataset

Performance	DSC[27]	XGBoost
Accuracy	0.99	0.999
Precision	**0.95**	0.879
Recall	0.67	**0.806**
F1	0.78	**0.841**

Finally, there is an LSTM approach for fraud detection based on the real dataset, which is covered in [1].

3 METHODOLOGY

3.1 PaySim Data

A considerable amount of research was done on a simulated scenario PaySim which can be found on Kaggle site[3]. In this dataset, the fraudulent behavior of the agents aims to profit by taking control of customers' accounts and trying to empty the funds by transferring them to another account and then cashing out of the system. We chose this dataset because it allows us to effectively compare the method to other studies working with the same dataset.

The data set contains 6.3 million transactions over one month of simulation, from which the number of fraudulent transactions is 8.2K, giving a ratio of circa 0.13%. Columns represent various attributes associated with transactions and numerical descriptions [min-max]:

- step - time step unit, one hour. [1-744]
- type - transaction category: CASH-IN 22%, CASH-OUT 36%, DEBIT 1%, PAYMENT 34% and TRANSFER 8%
- amount - the amount of the transaction in local currency.[0-92.5M]
- nameOrig - a customer who started the transaction. 6.35M unique values.
- oldbalanceOrg - customer's initial balance before the transaction. [0-59M]

[2] https://www.holisticai.com/blog/explainable-ai-dimensions
[3] https://www.kaggle.com/datasets/ealaxi/paysim1

- newbalanceOrig - customer's new balance after the transaction. [0-49.6M]
- nameDest - the recipient of the transaction, customers start C, merchants with M. 2.72M unique values, 79% M, 21% C
- oldbalanceDest - initial balance recipient before the transaction. There is no information for customers that start with M. [0-356.0M]
- newbalanceDest - new balance recipient after the transaction. There is no information for customers that start with M [0-356.1M].
- isFraud - These are the transactions made by the fraudulent agents inside the simulation.

3.1.1 Explorative Data Analysis. Figure 1 shows the fraudulent and valid transaction amount density distributions. Looking into the median and average values, it can be seen that both the Fraudulent and the Valid distributions are shifted with respect to each other. The average and median of fraudulent transactions are higher than the valid ones, meaning the transaction value should play a role in the fraud classification.

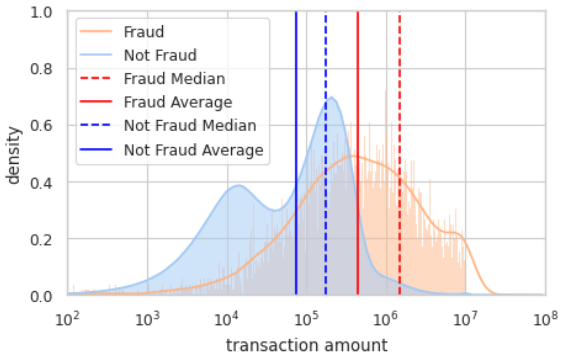


Figure 1: Fraudulent (Orange) and Valid (Blue) transaction density distributions. Dashed red and blue lines represent medians, and solid lines represent averages of fraudulent and valid transactions

Additional differences between the transactions are shown in the count plot in Figure 2. Here, it appears that only the TRANSFER and CASH_OUT transactions can be fraudulent.

Based on those figures, the expected Rule should be based on the transaction type, as *TRANSFER* or *CASH_OUT* to decide if a transaction is fraudulent, in addition to the transaction amount influence.

3.1.2 Feature engeneering. For the missing value treatment and aggregates calculation we followed the work of Visbeek et al. [27]. The SQL code is found in Appendix A.

Missing values treatment. No *null* or *NaN* values were identified in the dataset, however, there are transactions with zero values for old and new balances. Those are transactions from or to the Merchants, who are the Customers with an ID starting with *M*. The balances for these types of transactions were changed to be equal to the amount of a transaction. An additional *external origin*, and

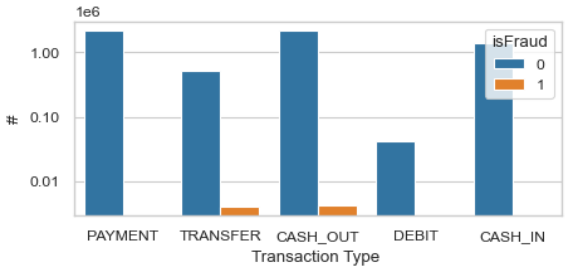


Figure 2: Fraudulent (Orange) and Valid (Blue) transaction count plot per type of transaction

external destination flags were calculated to mark those types of transactions. In Table 2 there is a schematic overview

Table 2: Missing values treatment for the transactions with zero balances before or after the transactions

Case	Flags
External origin	$old\_balance\_origine = amount$
Old balance origin is zero	$external\_orig\_flg = True$
External destination	$new\_balance\_destination = amount$
New balance destination	$external\_dest\_flg = True$

Test, train, validation. The transaction data was randomly split into train (85%) and test(15%) sets. Subsequently, the training set was split into the validation (15%) and train (85%) sets. Assuming the receiver is the fraudulent agent, the splits were performed on groups by the name of the destination.

Data scaling. The numerical attributes of the data set were scaled with a standard scaler fitted on the training dataset.

Aggregates calculation. We extend the datasets with additional features: For each name destination

- Average amount of the last 7 transactions including the current transaction
- Average amount last 3 transactions including the current transaction
- MAX amount of the last 7 transactions including the current transaction
- MAX amount of the last 3 transactions including the current transaction

3.2 Synthetic generated test data

Dummy Set. To understand the role of hyperparameters such as the number of inference steps and program templates, we first test our methods on a small set with dummy synthetic data, the dataset consisted of five binary columns A, B, C, D, and Target. Where the Target column is true if A, B, C, and D are True. The values for A, B, C, and D were randomly generated and are equal to *True* or *False*. The target predicate should be correctly learned and report 100% accuracy.

Additional Fraud scenarios. The final datasets were created to explore the ability of the method to model recursive predicates. These Fraud scenarios differ from the one described in PaySim, intended to create a rule that can define a chain of fraudulent events. The concept is based on the transitivity case and the graph connection examples from ∂ILP. The transitivity case was generated based on the random generation of the transactions between different entities, for the cases where one entity received a fraudulent transaction and then performed a transaction with another entity, this kind of transaction was defined as a Chain of Fraud. The examples of the datasets are presented in Appendix D.1.

3.3 Experimental Setup

The implementation of ∂ILP method is based on the code published by ai-systems [4]. The aggregated features were made with the DuckDb package [5] using SQL queries. The models based on PaySim and Dummy datasets were trained with an X13 Dell laptop, and the recursive examples were trained on the Snellius supercomputer [6].

3.4 ∂ILP

The state-of-the-art approach ∂ILP extends the modern machine learning paradigm by combining the advantages of ILP and the neural-network-based systems [10]. This work will focus on applying ∂ILP which offers an automatic rule derivation on the Fraud Detection.

The rules of an ILP framework are written as clauses of the following form:

$$H \leftarrow B_1, B_2, ..., B_n, \tag{2}$$

where atom H is defined as the head of the clause and the set of atoms $B_1, B_2, ..., B_n$ is defined as the body of the clause. Atom is defined as a predicate applied to a set of terms. Where each term is a variable, for example, a client ID. If all the atoms in the body are true, then the head is necessarily true. The clause is also defined as a definite clause because it has only one head.

For example, the following program defines the set of rules $\mathcal{R}$ for connected relations as the transitive closure of the edge relation:

$$\begin{aligned} &connected(X, Y) \leftarrow edge(X, Y) \\ &connected(X, Y) \leftarrow edge(X, Z), connected(Z, Y). \end{aligned} \tag{3}$$

Here, the *background knowledge* can be defined as a set of all edges, the *positive examples* as a set of all known connections, and the *negative examples* as a set of examples where the variables are not connected.

∂ILP learns a set $\mathcal{R}$ of definite clauses such that the union set of background facts $\mathcal{B}$ and $\mathcal{R}$ entails all the atoms in a set of positive examples $\mathcal{P}$, and does not entail all the atoms in a set of negative examples $\mathcal{N}$:

$$\begin{aligned} &\mathcal{B}, \mathcal{R} \vDash \gamma, \forall \gamma \in \mathcal{P} \\ &\mathcal{B}, \mathcal{R} \nvDash \gamma, \forall \gamma \in \mathcal{N} \end{aligned} \tag{4}$$

[4] https://github.com/ai-systems/DILP-Core
[5] https://duckdb.org/
[6] https://www.surf.nl/diensten/snellius-de-nationale-supercomputer

3.5 Clause Generations

Two types of predicates are distinguished in ∂ILP: the intensional and extensional predicates. The set of extensional predicates P_e are the given predicates from the background knowledge, and the set of intensional P_i are the predicates to be learned. The set P_i consists of the target predicate and additional auxiliary predicates P_a. The central component of ∂ILP is based on generating a list of possible definite clauses for intensional predicates, also known as a language bias [26]. The clauses are generated by the so-called rule template τ, which defines the range of clauses to generate. Two clauses define each predicate p, therefore there are two templates (τ_p^1, τ_p^2).

$$\tau = (n_\exists, int) \tag{5}$$

Where $n_\exists$ specifies the number of existentially quantified variables allowed in the clause, and int is a flag that determines whether the atoms in the clause can use intensional predicates P_i.

The following restrictions are applied when generating clauses:

- Each clause consists of exactly two atoms. Where an atom γ is any grounded predicate.
- A predicate has a maximal arity of two.
- The variable that appears in the head of the clause must appear in its body.
- An atom is not used in the same clause's head and body (circular restriction).

It was discovered during development, that the software suite generated rules of the following shape:

$$\begin{aligned} Fraud(X) &\leftarrow Predicate1(X) \\ Predicate1(X) &\leftarrow Fraud(X), Predicate2(X) \end{aligned} \tag{6}$$

The Atom $Fraud(X)$ does not appear directly in the body of the same rule but appears after in the body of a dependent Atom $Predicate1(X)$. Therefore, in addition to the formal restrictions mentioned in the work of Evans and Grefenstette [10], an extension of the circular restriction was introduced:

- The target predicate with the same variable can not appear in the head and body of both clauses defining the predicate.

3.6 Induction as Satisfiability

Considering $C_p^{i,j}$ to be j-th generated clause for τ_p^i template for an intensional predicate p, in a set $cl(\tau_p^i)$. The problem of finding the most suitable definition for a predicate P can be defined as a satisfiability problem, by introducing a Boolean set Φ consisting of a set of propositions each of a shape $(|cl(\tau_p^1)|, |cl(\tau_p^2)|)$ indicating which of the clauses in $C_p^{i,j}$ are to be used in the logical program for that predicate.

To find the relevant clauses ∂ILP uses a continuous relaxation of the truth value for every atom and then associates each pair of clauses with a weight that represents the probability of the pair being a part of the induced program. Then each ground atom γ can receive values in the range $[0, 1]$, not only 0 or 1. Similarly, to find a correct clause for the definition of an intensional predicate p in the set of intensional predicates P_i, the Boolean set Φ is converted to a set of continuous weights W to determine the clause $C_p^{i,j}$ is part of

a program for that predicate. W consists of a set of weight matrices $\mathbf{W}_p$, one matrix for each auxiliary predicate $p \in P_i$, in the same shape as $(|cl(\tau_p^1)|, |cl(\tau_p^2)|)$. Then the $softmax$ over $\mathbf{W}_p$ is applied to convert it to the probability of a clause $C_p^{i,j}$.

Two terms are introduced to perform the above procedure, the valuations and the function $\mathcal{F}_c$ over the valuations:

- A vector $\mathbf{a} = [0,1]^n$ based on the ground atom γ from the set G of n ground atoms is called *valuations*, and it represents the relaxed values of the ground atoms.
- For each clause c from set of clauses $C_p^{i,j}$ there is a function $\mathcal{F}_c$ over set of ground atoms G that maps valuations $[0,1]^n$ to another vector $[0,1]^n$ based on the definition of the clause. $\mathcal{F}_c(\mathbf{a}) : [0,1]^n \rightarrow [0,1]^n$. The values are calculated based on all combinations of grounded atoms in the clause's body that induce the clause's grounded head atom. In each combination, atoms are element-wise multiplied, and the maximum among the products equals $\mathcal{F}_c(\mathbf{a})$.

After applying $\mathcal{F}_c$ on each clause in $C_p^{j,k}$, the set $\mathcal{G}_p^{j,k}(\mathbf{a})$ is defined as follows:

$$\mathcal{G}_p^{j,k}(\mathbf{a}) = max(\mathcal{F}_p^{1,j}(\mathbf{a})[i], \mathcal{F}_p^{2,k}(\mathbf{a})[i]) \tag{7}$$

where $\mathbf{a}[i]$ is the i-th component of $\mathbf{a}$.

Having defined the above auxiliary function, the inference steps for the valuations $\mathbf{a}_t$, where t is an inference step, can be defined as follows:

$$\begin{aligned} \mathbf{c}_t^{p,j,k} &= \mathcal{G}_p^{j,k}(\mathbf{a}_t) \\ \mathbf{b}_t^p &= \sum_{j,k} \left(\mathbf{c}_t^{p,j,k} \cdot \frac{e^{\mathbf{W}_{p[j,k]}}}{\sum_{j',k'} e^{\mathbf{W}_{p[j',k']}}} \right) \\ \mathbf{a}_{t+1} &= f_{amalgamate}(\mathbf{a}_t, \sum_{p \in P_i} \mathbf{b}_t^p) \end{aligned} \tag{8}$$

$$f_{amalgamate}(x,y) = x + y - x \cdot y$$

with initial conditions on $\mathbf{a}$:

$$\mathbf{a}_0[i] = \begin{cases} 1 \text{ if } \gamma_i \in \mathcal{B} \\ 0 \text{ otherwise.} \end{cases} \tag{9}$$

Given the positive and the negative instances of the ground predicates sets $\mathcal{P}$, $\mathcal{N}$, ∂ILP ultimately becomes a binary classification problem, to match all the positive and negative instances. Therefore training is done by minimization of cross-entropy loss, commonly used by the neural network classifiers [25]:

$$\begin{aligned} loss = -\mathbb{E}_{\gamma,\lambda}[\lambda \cdot log(p(\lambda|\gamma, W, \Pi, \mathcal{B})) + \\ +(1-\lambda) \cdot log(1 - p(\lambda|\gamma, W, \Pi, \mathcal{B}))] \end{aligned} \tag{10}$$

[7]

where λ is a label of the ground atom γ,

$$\lambda = \begin{cases} 1 \text{ if } \gamma \in \mathcal{P} \\ 0 \text{ if } \gamma \in \mathcal{N}. \end{cases} \tag{11}$$

and Π is a program that defines the set of generated clauses.

[7]The original *loss* equation also contains the Language Frame $\mathcal{L}$ term, here it was ommitted for the simplicity.

$p(\lambda|\gamma, W, \Pi, \mathcal{B})$ is the conditional probability, of the predicted label λ, and it equals the evaluation $a[\gamma]$:

$$p(\lambda|\gamma, W, \mathcal{B}) = a[\gamma] \tag{12}$$

The loss minimization is solved by using stochastic gradient descent, by calculating the gradient of the loss function with respect to the rule weights set $\frac{\partial loss}{\partial W}$. Since Equation 12 is differentiable, based on the functions constructing it, the gradient is possible to calculate and to find the weight matrices W, which will determine the desirable clauses for each predicate.[8]

3.7 Pipeline

In this section, we describe different aspects of the implementation. The general overview of the process is described in Figure 3.

3.7.1 Adjustment of ∂ILP to use tabular data. ∂ILP is applied to background knowledge consisting of a set of facts, and positive and negative examples of the predicates. The set of facts is binary, therefore the existing input data should be adjusted to the same binary format. A work of Ciravegna et al. [6] about Logic Explained Networks (LEN), suggested discretizing numerical data into different bins to enable a neural learner to use the data. We exploit a similar approach for importing the fraudulent data into the model, by applying a threshold to values to test if it is above or below it. Thus converting a numerical column to a binary one. The binarised column is called a predicate of arity one. Because a predicate is a function of a variable, for the grounding purposes of a variable, it is assigned to an index of the transaction in the data set, thus explicitly applying the uniqueness of the relevant transaction, for example, $isFraud(X)$, is a predicate $isFraud$ of the transaction X. The DT and the DSC thresholds will be used for the binarisation purpose of the tabular data.

When discussing more complicated predicates with an arity of two, when considering a predicate based on sender and receiver, the grounding of the variable's facts is based on the sender and receiver identity number, assuming that the identity number is unique. For example, $isFraud(X, Y)$, is a predicate $isFraud$ of a transaction between X and Y.

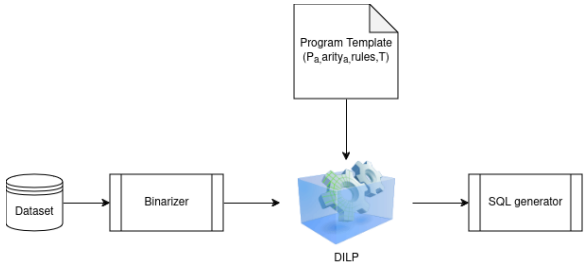


Figure 3: Pipeline ∂ILP. The dataset is any tabular dataset. Binarizer converts from numerical values to binary values. Program Template defines what set of clauses to generate. SQL generator converts the output rules to an SQL query

[8]For further details, we refer the reader to the original article [10].

3.7.2 Rule size. Rule size is defined through the number of possible predicate columns to incorporate in the logical program.

3.7.3 Program Template. As discussed before, ∂ILP requires an input of Program Template, consisting of the inference step T, a set of auxiliary predicates as in Rule size, and rule templates. The templates to generate the intensional and Target predicates are elaborated in Appendix C.1.

3.7.4 SQL Query. For the 1-arity predicate, an SQL query was generated based on the derived rule and applied to the tabular. An example rule with a form (Equation 13):

$$\begin{aligned} Target(X_0) &\leftarrow Pe_1(X_0), pred1(X_0) \\ Target(X_0) &\leftarrow Pe_2(X_0), pred2(X_0)(X_0) \\ pred1(X_0) &\leftarrow Pe_3(X_0), pred2(X_0) \\ pred2(X_0) &\leftarrow Pe_4(X_0). \end{aligned} \tag{13}$$

Following this rule, the SQL generator gives the query:

```
Select
    Pe4 as pred2,
    Pe3 as pred1,
    Pe2 and pred2 or Pe1 and pred1 as Target,
    from Fraud_Table
```

3.8 Evaluation metrics

3.8.1 Performance. The performance of the fraud detection framework will be evaluated by the commonly used metrics for classification: accuracy, recall, precision, and F1 score. Additionally, the MCC score will be evaluated [19]. MCC takes into account all four values of the confusion matrix, therefore it has a high score only if the classifier can correctly predict the majority of both positive and negative data instances [5]. It values as: −1 all predictions are wrong, 0 predictions are random, and +1 predictions are perfect. Due to the imbalanced data (see Section 3.1) the accuracy score is less relevant. Therefore, we focus mainly on precision, F1, and MCC scores.

$$\begin{aligned} accuracy &= \frac{tp+tn}{tp+tn+fp+fn} \\ precision &= \frac{tp}{tp+fp} \\ recall &= \frac{tp}{tp+fn} \\ F1 &= \frac{2 \cdot precision \cdot recall}{precision + recall} \\ MCC &= \frac{tp \cdot tn - fp \cdot fn}{\sqrt{(tp+fp)(tp+fn)(tn+fp)(tn+fn)}} \end{aligned} \tag{14}$$

4 RESULTS

The following sections present the results of rules learning over the generated data. In each case, the number of positive and negative instances is mentioned. Where the positive instances are where the Target Predicate is $True$, and the negative where the Target is $False$. The training set table may contain the rows where all the fact columns are $False$. Those are dropped since ∂ILP requires a set of facts that are $True$ over the constant sets. Therefore the sum of the instances may be less than the total number of the data rows.

Derived rules are presented as they appeared from the derivation, there are several cases where a head atom is defined by a duplication of an atom in the body of the rule. This is explained by the nature of the approach, which requires a body to be defined by exactly two atoms.

4.1 A, B, C, D scenario

The first experiment was to test the influence of the number of inference steps on ∂ILP performance and derived rules based on a dummy dataset. In Appendix E and Appendix F additional results of tests for *A and B* or *A, B, and C* are presented.

Table 3 summarises the A, B, C, and D experiment results. For the case of the 10 inference steps result without the Target recursion is reported

For the different inference steps, the following rules were achieved

Table 3: A, B, C, and D learning rules performance for different inference T steps, a fraction of positive target 7%, 7 positive and 86 negative examples out of $N = 100$ generated rows.

Performance	$T = 2$	$T = 3$	$T = 5$	$T = 10$
Train time [sec]	94	156	309	410
Accuracy	0.96	0.94	1	1
Precision	0.64	0.54	1	1
Recall	1	1	1	1
F1	0.778	0.7	1	1
MCC	0.78	0.71	1	1

$T = 2$ inference steps

$$\begin{aligned} Target(X_0) &\leftarrow D(X_0), pred2(X_0) \\ pred1(X_0) &\leftarrow D(X_0), pred2(X_0) \\ pred2(X_0) &\leftarrow C(X_0), A(X_0) \end{aligned} \tag{15}$$

Here the rule only partially covers the dataset, because when rephrased it is

$$Target(X_0) \leftarrow D(X_0), C(X_0), A(X_0) \tag{16}$$

remarkably $pred2$ does not influence the solution

$T = 3$ inference steps

$$\begin{aligned} Target(X_0) &\leftarrow pred1(X_0), pred1(X_0) \\ pred1(X_0) &\leftarrow pred2(X_0), B(X_0) \\ pred2(X_0) &\leftarrow C(X_0), D(X_0) \end{aligned} \tag{17}$$

Here the rule again partially covers the dataset, but with other predicates:

$$Target(X_0) \leftarrow B(X_0), C(X_0), D(X_0) \tag{18}$$

$T = 5$ inference steps

$$\begin{aligned} Target(X_0) &\leftarrow pred1(X_0), B(X_0) \\ pred1(X_0) &\leftarrow pred2(X_0), A(X_0) \\ pred2(X_0) &\leftarrow C(X_0), D(X_0) \end{aligned} \tag{19}$$

which can be rephrased as:

$$Target(X_0) \leftarrow B(X_0), A(X_0), C(X_0), D(X_0) \tag{20}$$

$T = 10$ Inference steps. This is an example of a circular dependency between $pred1$ and $Target$ predicates derived during training.

$$\begin{aligned} Target(X_0) &\leftarrow pred1(X_0), B(X_0) \\ pred1(X_0) &\leftarrow Target(X_0), Target(X_0) \\ pred2(X_0) &\leftarrow A(X_0), D(X_0) \end{aligned} \tag{21}$$

which can be rephrased as:

$$Target(X_0) \leftarrow B(X_0), Target(X_0), A(X_0), D(X_0) \tag{22}$$

meaning $Target$ depends on $Target$

$T = 10$ (Prevent recursion)

$$\begin{aligned} Target(X_0) &\leftarrow pred1(X_0), C(X_0) \\ pred1(X_0) &\leftarrow pred2(X_0), A(X_0) \\ pred2(X_0) &\leftarrow B(X_0), D(X_0) \end{aligned} \tag{23}$$

which is also a full rule, covering the dataset. But $T = 5$ is enough inference steps to cover such a dataset.

4.2 PaySim learning

Training on the original training set was impossible due to memory limitations. We trained ∂ILP on two smaller trainsets, consisting of 50:50% and 1% Fraud ratio.

4.2.1 Baseline performance. For establishing a baseline, a DT classifier and a classifier based on the DSC rule were evaluated (Appendix B). When the DT classifier was tuned to maximize the MCC score. The performances for the training and test sets as summarised in Table 4.

Table 4: Baseline Performance of DT, and DSC Rule

	DT		DSC	
Performance	training set	test set	training set	test set
Accuracy	0.999	0.999	0.999	0.999
Precision	0.969	0.971	0.981	0.984
Recall	0.647	0.665	0.493	0.501
F1	0.776	0.789	0.656	0.664
MCC	0.792	0.803	0.695	0.702

4.2.2 Parameters. All the tests were run with $T = 5$ amount of inference steps, based on the assumption, that the number of the required auxiliary predicates is not greater than 2 predicates in the "A, B, C, and D" simulated case.

4.2.3 DT thresholds. In Table 5 the results for the DT thresholds dataset case are summarized,

Derived rules for $|p_a| = 1$:

$$\begin{aligned} isFraud(X_0) &\leftarrow external\_dest(X_0), pred1(X_0) \\ pred1(X_0) &\leftarrow amount > 1.297 \end{aligned} \tag{24}$$

Table 5: DT thresholds predicate-based learning rules performance for one and two auxiliary predicates, a fraction of positive Fraud instances 50%, 100 positive and 100 negative examples out of $N = 200$ rows.

	$\|p_a\| = 1$		$\|p_a\| = 2$	
Performance	training set	test set	training set	test set
Train time [sec]	157	-	275	-
Accuracy	0.540	0.999	0.540	0.999
Precision	1.000	1.000	1.000	1.000
Recall	0.08	0.176	0.080	0.176
F1	0.148	0.299	0.148	0.299
MCC	0.204	0.419	0.204	0.419

Derived rules for $|p_a| = 2$:

$$\begin{aligned} isFraud(X_0) &\leftarrow external\_dest(X_0), pred2(X_0) \\ pred1(X_0) &\leftarrow pred2(X_0), external\_dest(X_0) \\ pred2(X_0) &\leftarrow amount > 1.297 \end{aligned} \tag{25}$$

both rules express the same rule from the form:

$$isFraud(X_0) \leftarrow external\_dest(X_0), amount > 1.297 \tag{26}$$

This led to a high *Precision* score on the test set, meaning there were no False Positives, i.e. *False Fraudulent* transactions, but with high underperforming on the rest of the metrics, resulting in a high number of False Negatives, i.e. a large number of *Fraudulent* transactions were not detected.

4.2.4 Dataset based on DSC rule. Results are presented in the Table 6.

Table 6: DSC threshold based learning rules performance for 1 and 2 auxiliary predicates, a fraction of positive Fraud instances 50%, 100 positive and 100 negative examples out of $N = 200$ rows.

	$\|p_a\| = 1$		$\|p_a\| = 2$	
Performance	training set	test set	training set	test set
Train time [sec]	305	-	558	-
Accuracy	0.715	0.9993	0.715	0.9993
Precision	1	0.974	1	0.974
Recall	0.43	0.501	0.43	0.501
F1	0.601	0.662	0.601	0.662
MCC	0.523	0.698	0.523	0.698

Derived rules for $|p_a| = 1$:

$$\begin{aligned} isFraud(X_0) &\leftarrow pred1(X_0), pred1(X_0) \\ pred1(X_0) &\leftarrow type\_TRANSFER(X_0), external\_dest(X_0) \end{aligned} \tag{27}$$

Derived rules for $|p_a| = 2$

$$\begin{aligned} isFraud(X_0) &\leftarrow pred2(X_0), pred2(X_0) \\ pred1(X_0) &\leftarrow pred2(X_0), external\_dest(X_0) \\ pred2(X_0) &\leftarrow type\_TRANSFER(X_0), \\ &\qquad external\_dest(X_0) \end{aligned} \tag{28}$$

for both cases the same rule was learned: $isFraud(X_0)$ if $type\_TRANSFER(X_0)$ and $external\_dest(X_0)$ are $True$, which explains the same performance results.

4.2.5 training set for 1% fraction fraud cases. To estimate the influence of a smaller fraction of the Fraudulent class on the training performance, we extracted 1000 non-fraudulent and 10 fraudulent transactions from DT and DSC thresholds datasets. In Table 7 the results for both cases are summarized, the rules were derived for 2 auxiliary predicates

Table 7: DT and DSC thresholds dataset learning rules performance for two auxiliary predicates, a fraction of positive Fraud instances 1%, 10 positive and 1000 negative examples out of $N = 1010$ rows.

	DT		DSC	
Performance	training set	test set	training set	test set
Train time [sec]	505	-	859	-
Accuracy	0.996	0.999	0.996	0.999
Precision	1.000	0.974	1.000	0.973
Recall	0.600	0.501	0.600	0.501
F1	0.750	0.662	0.75	0.662
MCC	0.773	0.698	0.773	0.698

Derived rules for DT thresholds dataset:

$$\begin{aligned} isFraud(X_0) &\leftarrow pred1(X_0), pred2(X_0) \\ pred1(X_0) &\leftarrow pred2(X_0), external\_dest(X_0) \\ pred2(X_0) &\leftarrow external\_dest(X_0), type\_TRANSFER(X_0) \end{aligned} \tag{29}$$

Derived rules for DSC thresholds dataset:

$$\begin{aligned} isFraud(X_0) &\leftarrow type\_TRANSFER(X_0), pred1(X_0) \\ pred1(X_0) &\leftarrow pred2(X_0), external\_dest(X_0) \\ pred2(X_0) &\leftarrow type\_TRANSFER(X_0), type\_TRANSFER(X_0) \end{aligned} \tag{30}$$

Remarkably, the rule derived on the DT data set is the same as for the DSC dataset one and it boils down to $isFraud(X_0) \leftarrow external\_dest(X_0), type\_TRANSFER(X_0)$, therefore the same performance.

4.2.6 Dataset based on DT thresholds including the negation. When deriving binary predicates based on the DT thresholds, the negated branches are not being checked by ∂ILP, because it cannot generate the negated predicates. Therefore additional predicates were generated, equaled to the negation of the first set. The Fraud templates should support the DT paradigm, which defines rules as *if A then B, else if not A then C*. The test was executed with two auxiliary predicates set up on both the 50% and the 1% fraud data set.

Derived rules for 50%:

$$\begin{aligned} isFraud(X_0) &\leftarrow NOT\{oldbalanceDest > -0.007\}(X_0), pred2(X_0) \\ isFraud(X_0) &\leftarrow pred2(X_0), amount > 1.297(X_0) \\ pred1(X_0) &\leftarrow NOT\{oldbalanceDest > -0.007\}(X_0), pred2(X_0) \\ pred2(X_0) &\leftarrow external\_dest(X_0), type\_TRANSFER(X_0) \end{aligned} \tag{31}$$

Table 8: DT thresholds dataset based learning rules performance for two auxiliary predicates, a fraction of positive Fraud instances 1%, 10 positive and 1000 negative examples out of $N = 1010$ rows.

	50 %		1 %	
Performance	training set	test set	training set	test set
Train time [sec]	1181	-	1958	-
Accuracy	0.715	0.999	0.996	0.999
Precision	1.000	0.974	1.000	0.973
Recall	0.430	0.501	0.600	0.501
F1	0.601	0.662	0.75	0.662
MCC	0.523	0.698	0.773	0.698

Derived rules for 1%:

$$\begin{aligned} isFraud(X_0) &\leftarrow type\_TRANSFER(X_0), pred1(X_0) \\ isFraud(X_0) &\leftarrow pred1, pred2(X_0) \\ pred1(X_0) &\leftarrow external\_dest(X_0), pred2(X_0) \\ pred2(X_0) &\leftarrow external\_dest(X_0), type\_TRANSFER(X_0) \end{aligned} \tag{32}$$

Also in this case the performance is similar to the DSC thresholds-based case, in a deeper analysis, only the

$$isFraud(X_0) \leftarrow external\_dest(X_0), type\_TRANSFER(X_0)$$

played a role, meaning the other rules were not engaged or also were $True$ when the main condition was $True$.

4.2.7 Applying a large number of predicates. An additional test was done with varying numbers of auxiliary predicates from two to eight to cover all possible columns in the DT dataset, which resulted in the same performance as reported for two predicates. Meaning it did not show a difference in performance.

4.3 Learning the Recursive Structures

The final set of experiments aimed to test the ability of ∂ILP to model recursive predicates. The Fraud scenario here is different than the one described in PaySim. The knowledge of Fraud is already known, for example by the superior classification algorithm, but there is an intention to derive patterns in the data.

4.3.1 Fraud Relationship. In this scenario, the intention was to derive based on the dataset a rule, that can find a fraudulent relationship, based on background knowledge. The tabular data set was prepared based on the example of Graph Connectedness from the original paper, and described in the extension in Appendix D.1. All the transactions are fraudulent.

The derivation of a rule took 957 [sec], based on the input consisting of 4 Facts and 9 Positive examples:

$$\begin{aligned} Fraudsters(X, Y) &\leftarrow Fraud(X, Y) \\ Fraudsters(X, Y) &\leftarrow Fraud(Z, Y), \\ &\qquad Fraudsters(Z, X) \end{aligned} \tag{33}$$

Here the first Rule is $Fraudsters(X, Y) \leftarrow Fraud(X, Y)$ is inherent based on the data set, as for the transaction between 1 and 2 ($Fraud(1, 2)$). The second rule is reflected by the example of Fraud

transactions between 1 and 2 ($Fraud(1,2)$) and 1 and 4 are Fraudsters from the facts ($Fraudsters(1,4)$), therefore it explains 2 and 4 are Fraudsters ($Fraudsters(2,4)$). Although it does appear in the facts, here we can see that it can be generalized to the examples which do not appear in the training set.

4.3.2 Chain of Fraud. This is an extension of the previous example, with a rationale to create a dataset with a chain of events, to find a rule for the transaction between three parties, which participated in Fraud. The dataset is presented in Appendix D.2.

The derivation of a rule took 1355 [sec], based on the input consisting of 36 Facts, 5 Positive examples, and 21 Negative examples:

$$Fraud\_Chain(X,Y) \leftarrow Fraud(Z,X), Transaction(X,Y) \quad (34)$$

That rule can be translated as *"There is a chain of Fraud between X and Y if there is a transaction from X to Y and a fraud event between any Z and X.*

There is a fraudulent transaction #8 from customer 16051 to customer 16086 and also a regular transaction #4 from customer 16086 to customer 16014, which satisfies the $Fraud(16051, 16086)$ and $Transaction(16086, 16014)$, and hence satisfies the head of the rule: $Fraud\_Chain(16086, 16014)$, see Figure 4

	orig	destination	Fraud	Fraud_chain--orig--destination
4	16086	16014	True	True
8	16051	16086	True	False

Figure 4: Fraud chain example from Appendix D.2

5 DISCUSSION

5.1 Performance

5.1.1 Dummy dataset. For the unit test case, when learning the rules from errorless dataset ∂ILP succeeds in finding the logical rules. The number of inference steps can be seen as a hyperparameter for tuning as in the case of A, B, C, D rule learning.

5.1.2 PaySim dataset. When applied to the PaySim dataset, regardless of the way of training on the DT or DSC converted Datasets, ∂ILP was in line with the performance on the test set in terms of all the metrics compared to the performance of the DSC approach. The DT Classifier has about 10% higher performance than ∂ILP achieved in terms of F1, Recall, and MCC metrics, as shown in Table 9. The original performance of DSC[27] is higher than the performance achieved by applying the rule on the data set in Recall and F1 and actually resembles the performance achieved by applying DT, which can be a result of the different dataset splits.

5.1.3 Data Conversion. We covered two approaches to utilizing tabular data for ILP methods, based on transaction IDs, or based on agent IDs. We showed how to prepare binary data based on the DT or DSC thresholds, and what Program Templates to use. When ∂ILP was applied to the dataset prepared by DT thresholds, the number of auxiliary predicates, and a fraction of Fraud influenced the performance. We saw that for data including only positive (non-negated) columns and 50-50% split, the performance based on the Recall, F1, and MCC were lower than for the rest of the approaches (0.176, 0.299. 0.419) vs (0.501, 0.662. 0.698), see Table 5 and 8. Therefore it is safe to claim that the general approach to converting a data set is by including both negated and non-negated columns in terms of above and below a specific threshold and defining a template for a target predicate rule that can combine both. In this way, the created dataset covers both possibilities. Finally, the dataset has to reflect a real ratio of the Target predicate in the original dataset.

Table 9: Performance comparison between DT, DSC and ∂ILP

Performance	∂ILP	DT	DSC	DSC[27]	XGBoost
Accuracy	0.999	0.999	0.999	0.99	0.999
Precision	0.973	0.971	**0.984**	0.95	0.879
Recall	0.501	0.665	0.501	0.67	**0.806**
F1	0.662	0.789	0.664	0.78	**0.841**
MCC	0.698	**0.803**	0.702	-	-

5.1.4 Recursion and Connectivity. ∂ILP succeeded in learning the rule for the Fraud Relationship and the Chain of Fraud cases.

5.2 Scalability

Evans and Grefenstette [10] explains the sources of memory consumption, The memory size depends on the set of invented predicates and constants size. The constant size is equal to the dataset length in our case.

Memory size also depends on the amount of the generated clauses. In the case of templates requiring the invention of the predicates with an arity of two, ∂ILP generates a very large dataset of clauses. This is the explanation for the large training time when applied to relatively small datasets in the case of connectivity and recursion.

5.3 Circular Dependency

∂ILP showed that there is a chance of creating rules that have circular dependencies between predicates, the chance is higher in the case of one arity predicates. We implemented the removal of circular dependency on the target predicate. Yet we suspect that because there is no restriction to using two head Atoms from two different clauses in each other bodies, there still is a chance of circularity in other predicates. How to implement additional circular restrictions on the predicate generation could be another future topic to investigate.

6 CONCLUSION

6.1 Summary

First, RQ2, RQ3, and RQ4 are going to be reflected, and then as a consequence RQ1. With regards to the RQ2, it was shown that when training on a small part of the PaySim dataset, ∂ILP cannot outperform the techniques used for data transformation to the Boolean format, required by ∂ILP. However, ∂ILP showed the ability to provide a more explainable rule than the DT, when examining the size of the formula. Training of the full PaySim dataset was not

possible due to the memory limitation, therefore it is not clear what would be the performance when trained on a larger training set. Therefore RQ2 is only partially answered when addressed on the PaySim dataset.

Concerning RQ3, dependency of performance of the rule size in terms of the number of used auxiliary predicates, this research question could not be fully answered. It is safe to say that this parameter can play the role of underfitting in the worst-case scenario as a Rule will not cover all the relationships in the data. In addition, as shown in the case when applying a varying number of rules, the performance stayed the same when learning the DT dataset (4.2.7).

Finally, with regards to RQ4, for more complicated scenarios that require inventing rules with Recursion Structures, ∂ILP successfully provided explanatory rules to the dataset that the methods such as DT, or DSC by the definitions of the approaches, are not able to derive.

Based on the RQ2, RQ3, and RQ4 - RQ1 can only be partially answered: ∂ILP can provide explanations for fraudulent datasets, but for limited amounts of data, it fails to perform on larger sets. For scenarios such as the chain of fraud events and fraudulent relationships, ∂ILP provided explanations that the classical method can not provide.

6.2 Limitations and Future Work

To the best of our knowledge, this is the first time the ILP approach with the help of ∂ILP applied to explain a Fraud dataset. Currently, the method seems insufficient for application in real-world scenarios; a key limitation is the usage of binary predicates, requiring the creation of the binary features from the numerical data with the help of other classification techniques. An additional overhead comes from the high complexity and memory consumption required to create predicates for the approach. Finally, expert knowledge is needed to define the templates for intensional predicate generation. The memory consumption and the template definition limitations were also discussed in the original work of [10].

Apart from the mentioned limitations, this work showed the potential of using the ILP paradigm for creating rules to describe fraudulent data, therefore the ILP methods deserve attention to be further developed and hybridized with other existing methods if necessary. One of the possible future work directions is to apply another neurosymbolic extension for ILP to the scenarios covered in this thesis such as [8, 21].

ACKNOWLEDGEMENT

I want to express my sincere gratitude to Dr. Erman Acar for his advice and patience, providing me with guidance throughout this research project. I am also grateful to Mara Smeets and my brother, Eugine, for proofreading the report and giving constructive feedback. Lastly, I want to thank my family and friends for their support during my studies.

REFERENCES

[1] Yara Alghofaili, Albatul Albattah, and Murad A Rassam. 2020. A financial fraud detection model based on LSTM deep learning technique. *Journal of Applied Security Research* 15, 4 (2020), 498–516.

[2] Yoshua Bengio, Olivier Delalleau, and Clarence Simard. 2010. Decision trees do not generalize to new variations. *Computational Intelligence* 26, 4 (2010), 449–467.

[3] Davide Beretta, Stefania Monica, and Federico Bergenti. 2022. A Comparative Study of Three Neural-Symbolic Approaches to Inductive Logic Programming. In *International Conference on Logic Programming and Nonmonotonic Reasoning*. Springer, 56–61.

[4] Leo Breiman, Jerome Friedman, Richard Olshen, and Charles Stone. 1984. Cart. *Classification and regression trees* (1984).

[5] Davide Chicco and Giuseppe Jurman. 2020. The advantages of the Matthews correlation coefficient (MCC) over F1 score and accuracy in binary classification evaluation. *BMC genomics* 21 (2020), 1–13.

[6] Gabriele Ciravegna, Pietro Barbiero, Francesco Giannini, Marco Gori, Pietro Lió, Marco Maggini, and Stefano Melacci. 2023. Logic explained networks. *Artificial Intelligence* 314 (2023), 103822.

[7] Marine Collery, Philippe Bonnard, François Fages, and Remy Kusters. 2022. Neural-based classification rule learning for sequential data. In *The Eleventh International Conference on Learning Representations*.

[8] Wang-Zhou Dai and Stephen H Muggleton. 2020. Abductive knowledge induction from raw data. *arXiv preprint arXiv:2010.03514* (2020).

[9] Andrzej Dudek, Marcin Pełka, et al. 2022. Symbolic data analysis as a tool for credit fraud detection. *Bank i Kredyt* 53, 6 (2022), 587–604.

[10] Richard Evans and Edward Grefenstette. 2018. Learning explanatory rules from noisy data. *Journal of Artificial Intelligence Research* 61 (2018), 1–64.

[11] UK Finance. 2023. ANNUAL FRAUD REPORT 2022. https://www.ukfinance.org.uk/policy-and-guidance/reports-and-publications/annual-fraud-report-2023.

[12] Oscar M Granados and Andrés Vargas. 2022. The geometry of suspicious money laundering activities in financial networks. *EPJ Data Science* 11, 1 (2022), 6.

[13] Petr Hajek, Mohammad Zoynul Abedin, and Uthayasankar Sivarajah. 2023. Fraud detection in mobile payment systems using an XGBoost-based framework. *Information Systems Frontiers* 25, 5 (2023), 1985–2003.

[14] Pascal Hitzler and Md Kamruzzaman Sarker. 2022. Neuro-symbolic artificial intelligence: The state of the art. (2022).

[15] Steven Kemp, David Buil-Gil, Asier Moneva, Fernando Miró-Llinares, and Nacho Díaz-Castaño. 2021. Empty streets, busy internet: A time-series analysis of cybercrime and fraud trends during COVID-19. *Journal of Contemporary Criminal Justice* 37, 4 (2021), 480–501.

[16] Remy Kusters, Yusik Kim, Marine Collery, Christian de Sainte Marie, and Shubham Gupta. 2022. Differentiable Rule Induction with Learned Relational Features. *arXiv preprint arXiv:2201.06515* (2022).

[17] J Kwik, T van Engers, et al. 2023. Performance or Explainability? A Law of Armed Conflict Perspective. (2023).

[18] Edgar Lopez-Rojas, Ahmad Elmir, and Stefan Axelsson. 2016. PaySim: A financial mobile money simulator for fraud detection. In *28th European Modeling and Simulation Symposium, EMSS, Larnaca*. Dime University of Genoa, 249–255.

[19] Brian W Matthews. 1975. Comparison of the predicted and observed secondary structure of T4 phage lysozyme. *Biochimica et Biophysica Acta (BBA)-Protein Structure* 405, 2 (1975), 442–451.

[20] Stephen Muggleton. 1991. Inductive logic programming. *New generation computing* 8 (1991), 295–318.

[21] Ali Payani and Faramarz Fekri. 2019. Inductive logic programming via differentiable deep neural logic networks. *arXiv preprint arXiv:1906.03523* (2019).

[22] Brenden K Petersen, Mikel Landajuela, T Nathan Mundhenk, Claudio P Santiago, Soo K Kim, and Joanne T Kim. 2019. Deep symbolic regression: Recovering mathematical expressions from data via risk-seeking policy gradients. *arXiv preprint arXiv:1912.04871* (2019).

[23] José Pombal, André F Cruz, João Bravo, Pedro Saleiro, Mário AT Figueiredo, and Pedro Bizarro. 2022. Understanding Unfairness in Fraud Detection through Model and Data Bias Interactions. *arXiv preprint arXiv:2207.06273* (2022).

[24] Litao Qiao, Weijia Wang, and Bill Lin. 2021. Learning accurate and interpretable decision rule sets from neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 35. 4303–4311.

[25] Usha Ruby and Vamsidhar Yendapalli. 2020. Binary cross entropy with deep learning technique for image classification. *Int. J. Adv. Trends Comput. Sci. Eng* 9, 10 (2020).

[26] Birgit Tausend. 1994. Representing biases for inductive logic programming. In *European Conference on Machine Learning*. Springer, 427–430.

[27] Samantha Visbeek, Erman Acar, and Floris den Hengst. [n. d.]. Explainable Fraud Detection with Deep Symbolic Classification. *methods* 2, 10 ([n. d.]), 17–28.

Appendix A FEATURE ENGINEERING SQL

Creating external destination and external origin flags,

```
SELECT *,
-- evaluating external accounts
    oldbalanceOrg==0 and newbalanceOrig==0 as external_orig,
    oldbalanceDest==0 and newbalanceDest==0 as external_dest,
-- Imputing zero values for external accounts
    CASE WHEN external_orig==True
        THEN  amount
        ELSE oldbalanceOrg
    END AS oldbalanceOrg_imputed,
    CASE WHEN external_dest==True
        THEN  amount
        ELSE newbalanceDest
    END AS newbalanceDest_imputed,
FROM fraud_tbl
```

evaluating aggregates

```
SELECT *,
-- calculate aggregetions (Average and Max) for last 7 including the current row for each name Destination
        AVG(amount) OVER (PARTITION BY nameDest ORDER BY index ROWS BETWEEN 6 PRECEDING AND CURRENT ROW) AS mean_last_7,
        MAX(amount) OVER (PARTITION BY nameDest ORDER BY index ROWS BETWEEN 6 PRECEDING AND CURRENT ROW) AS max_last_7,

-- calculate aggregetions (Average and Max) for last 3 including the current row for each name Destination
        AVG(amount) OVER (PARTITION BY nameDest ORDER BY index ROWS BETWEEN 2 PRECEDING AND CURRENT ROW) AS mean_last_3,
        MAX(amount) OVER (PARTITION BY nameDest ORDER BY index ROWS BETWEEN 2 PRECEDING AND CURRENT ROW) AS max_last_3,

-- deviation from the aggregated values
            amount-mean_last_7 as deviation_from_mean_7_days,
            amount-mean_last_3 as deviation_from_mean_3_days,
            amount-max_last_7 as deviation_from_max_7_days,
            amount-max_last_3 as deviation_from_max_3_days,

        FROM
            data
            ORDER BY index
```

Appendix B DT AND DSC DATASETS PREPARATION

The DT classifier trained on the training dataset, with hyperparameter optimization on the development dataset by cross-validated grid-search over a set of leaf nodes with target score based on MCC score, resulting in a tree with $max\_leaf\_nodes = 8$ parameter see Figure 5. The boundaries of the derived rules were converted to the predicate columns, such as a column is $True$ if the related value is equal to or greater than the derived decision boundary. Resulting in a dataset with the amount of columns equal to the amount of the decision boundaries.

$$\begin{aligned} &type == TRANSFER \\ &external\_dest == True \\ &oldbalanceDest > -0.007 \\ &amount > 1.297 \\ &amount > 3.079 \end{aligned} \tag{35}$$

For the conversion dataset based on the DSC rule, we took the rule from the work of [27]:

$$Fraud \leftarrow (type == TRANSFER), externalDest, (amount - maxDest7 > -0.15) \tag{36}$$

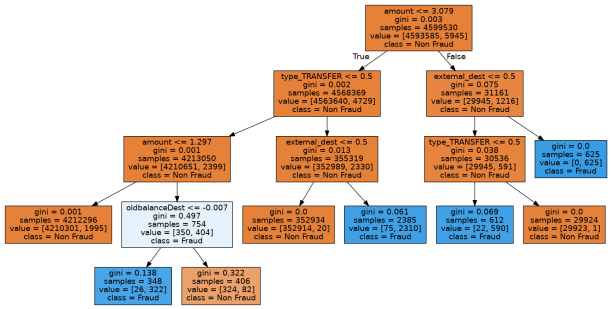


Figure 5: Derived Decision Tree for Fraud Detection, left branch is a *True* case, right branch is a *False* case

The following predicates were used, including the whole *type* possibilities set:

$$\begin{aligned} deviation\_from\_max\_7 = amount - maxDest7 > -0.15 \\ external\_dest == True \\ type == CASH\_IN \\ type == CASH\_OUT \\ type == DEBIT \\ type == PAYMENT \\ type == TRANSFER \end{aligned} \tag{37}$$

Appendix C TEMPLATES FOR AUXILIARY PREDICATE GENERATION

When discussing templates, there is a distinction between the different types of rules the system is expected to provide. The first is without recursion as in the case of the Unit test datasets and Paysim. The second one is with recursion, as for the cases of Fraud relationship and Fraud chain event.

C.1 No Recursion

The templates used in this study are divided between, what kind of rules are expected to be derived. For the case where the expected rule form is *A is true if B and C and D ... are true*, the number of generated auxiliary predicates equals the set size of the extensional predicates less 2. Each intensional predicate template allows one intensional predicate in the body, whereas the last predicate has zero intensional predicates in its body. Each of the intensional predicates has only one rule in its template. In the example of *A, B, C, and D* case, the number of extensional predicates is 4, meaning the amount of generated predicates is 2, and the derived template is shown at 38:

$$\begin{aligned} \tau^1_{target} &= (n_\exists = 0, int = 1) \\ \tau^2_{target} &= None \\ \tau^1_{p1} &= (n_\exists = 0, int = 1) \\ \tau^2_{p1} &= None \\ \tau^1_{p2} &= (n_\exists = 0, int = 0) \\ \tau^2_{p2} &= None \end{aligned} \tag{38}$$

For the case where the expected rule form is *A is true IF* X_1 *and* X_2 *and* X_3 *... are true OR* Y_1 *and* Y_2 *and* Y_3*.... are true* , as in Decision Trees with negated columns, the Target appreciate has two rules in the definition to allow the formulation of *OR*, the rest remains the same as shown in 41

$$
\begin{aligned}
\tau^1_{target} &= (n_\exists = 0, int = 1) \\
\tau^2_{target} &= (n_\exists = 0, int = 1) \\
\tau^1_{p1} &= (n_\exists = 0, int = 1) \\
\tau^2_{p1} &= None \\
\tau^1_{p2} &= (n_\exists = 0, int = 0) \\
\tau^2_{p2} &= None
\end{aligned}
\tag{39}
$$

C.2 Fraud Relationship

Here the template for the first clause for the Target predicate does not allow the usage of an existential variable but allows to use of an intensional predicate, i.e. the Target predicate. The second clause allows usage of the existential variable.

$$
\begin{aligned}
\tau^1_{target} &= (n_\exists = 0, int = 1) \\
\tau^2_{target} &= (n_\exists = 1, int = 1)
\end{aligned}
\tag{40}
$$

C.3 Fraud Chain

The first clause for the Target predicate does allow the usage of an existential variable but does not allow the generation of an intensional predicate, i.e. the Target predicate is expected to be defined only based on the extensional predicates. The second clause is Null.

$$
\tau^1_{target} = (n_\exists = 1, int = 0) \tag{41}
$$

Appendix D ADDITIONAL FRAUD SCENARIOS

D.1 Fraud Relationship

The Fraud relationship dataset is presented in Figure 6. It is built from facts and positive examples described below and then fed to a build pipeline. The set of constants a, b, c, d is equal to $1, 2, 3, 4$

$$facts = \{Fraud(a, b), Fraud(b, c), Fraud(c, d), Fraud(b, a))\} \tag{42}$$

where $Fraud(x, y)$ is means that there is a direct transaction from x to y that is Fraudulent.

$$\begin{aligned} \mathcal{P} = \{&Fraudsters(a, b), Fraudsters(b, c), Fraudsters(c, d), Fraudsters(b, a), Fraudsters(a, c), \\ &Fraudsters(a, d), Fraudsters(a, a), Fraudsters(b, d), Fraudsters(b, a), Fraudsters(b, b)\} \\ \mathcal{N} = \{\} \end{aligned} \tag{43}$$

Target Predicate is $Fraudsters(x, y)$, which is $True$, if there is any fraudulent relationship between x, and y. For example, Agent 1 is a fraudster from the positive examples set, indicated by $Fraudsters(1, 1)$. From the fact set there is a transaction from 1 to 2 therefore there is a fraudulent relationship between both, further, there is a transaction between 2 and 3, therefore those two are also fraudulent agents. This is why 1 and 3 are also fraudulent agents, although there is no direct transaction between them.

	X	Y	Fraudsters--X--Y	Fraud--X--Y
0	1	2	True	True
1	2	3	True	True
2	3	4	True	True
3	2	1	True	True
4	1	3	True	False
5	1	4	True	False
6	1	1	True	False
7	2	4	True	False
8	2	2	True	False

Figure 6: Fraud relationship dataset

D.2 Chain of Fraud

The Chain of Fraud dataset is presented in Figure 7. This dataset represents a scenario where there is a transaction between some X customer and Y customer, some transactions are randomly labeled as Fraud transactions. In addition, if there exists a transaction between Y and Z, and Y was a recipient in a fraudulent transaction, then the transaction between Y and Z is assigned as both Fraud and Fraud Chain.

	orig	destination	Fraud	Fraud_chain--orig--destination
0	16058	16066	False	False
1	16065	16052	False	False
2	16036	16067	True	False
4	16086	16014	True	True
5	16043	16004	True	False
6	16011	16067	False	True
7	16002	16011	True	False
8	16051	16086	True	False
9	16080	16077	False	False
10	16032	16075	False	True
11	16054	16056	False	False
13	16038	16024	True	False
14	16019	16029	False	False
15	16046	16021	False	False
16	16042	16025	False	False
17	16056	16080	False	False
19	16077	16061	True	False
21	16024	16033	True	True
22	16002	16032	True	False
23	16003	16070	False	False
24	16013	16085	False	True
25	16040	16031	False	False
26	16072	16013	True	False
27	16019	16071	False	False
28	16072	16056	False	False
29	16026	16024	False	False

Figure 7: Chain of Fraud dataset, *orig* **represents code for a customer code that performed the transaction,** *destination* **represents a customer code that received the transaction.** *Fraud* **is a flag if a transaction was fraudulent,** $Fraud\_Chain--orig--destination$ **is a flag if a transaction was in a chain of fraud event**

Appendix E A AND B LEARNING

For all the runs the the following Rule was derived

$$Target(X_0) \leftarrow B(X_0), A(X_0) \tag{44}$$

Table 10 summarises the A and B experiments results.

Table 10: A, B learning rules performance for different inference T steps, fraction of positive target 22%, 22 positive and 52 negative examples out of $N = 100$ generated rows

Performance	$T = 2$	$T = 3$	$T = 5$	$T = 10$
Training time [sec]	19	29	49	95
Accuracy	1	1	1	1
Precision	1	1	1	1
Recall	1	1	1	1
F1	1	1	1	1
MCC	1	1	1	1

Appendix F A, B, AND C LEARNING

Table 11 summarises the A, B, and C experiment results.

For the different inference steps, the following rules were achieved

$T = 2$

$$\begin{aligned} Target(X_0) &\leftarrow pred1(X_0), pred1(X_0) \\ pred1(X_0) &\leftarrow B(X_0), C(X_0) \end{aligned} \tag{45}$$

$T = 3, T = 5$

$$\begin{aligned} Target(X_0) &\leftarrow pred1(X_0), C(X_0) \\ pred1(X_0) &\leftarrow A(X_0), B(X_0) \end{aligned} \tag{46}$$

$T = 10$

$$\begin{aligned} Target(X_0) &\leftarrow B(X_0), pred1(X_0) \\ pred1(X_0) &\leftarrow A(X_0), C(X_0) \end{aligned} \tag{47}$$

Table 11: A, B, and C learning rules performance for different inference T steps, a fraction of positive target 8%, 8 positive and 83 negative examples out of $N = 100$ generated rows

Performance	$T = 2$	$T = 3$	$T = 5$	$T = 10$
Training time [sec]	43	62	120	250
Accuracy	0.92	1	1	1
Precision	0.5	1	1	1
Recall	1	1	1	1
F1	0.67	1	1	1
MCC	0.68	1	1	1