Differentiable Inductive Logic Programming (∂ILP) for Fraud Detection

Data Science Master's Thesis - 2024

Boris Wolfson, Dr. Erman Acar [†]

September 10, 2024

University of Amsterdam

[†] Institute for Logic, Language and Computation

Code

Github repo: *https://github.com/wolfbrr/ThesisDS*

Amsterdam
Data Science

OUTLINE

1. Introduction
2. ∂ILP
3. Fraud Detection Framework
4. Summary

Introduction

Introduction: Logic Programming - Definitions

Logic programming

- Family of programming languages in which the central component is an if-then rule/clause
- The rules of an ILP framework are written as a set of definite clauses of the following form:

$$\alpha \leftarrow \alpha_1, \alpha_2, ..., \alpha_n, \tag{1}$$

Here:

- A set of atoms $\{\alpha_i\}$ defines a body of the rule.
- α is the head atom of the rule, and equals to True, if all the atoms in the body are True
- Each atom is defined by a predicate

Introduction: Logic Programming-Examples

Example - predicates:

- Q(X,Y): Q is a predicate of arity 2, can be ”x is greater than y” and ”x is the father of y”, or x transferred money to y

Example - Logic Program

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

$$\begin{aligned} connected(X,Y) &\leftarrow edge(X,Y) \\ connected(X,Y) &\leftarrow edge(X,Z), connected(Z,Y). \end{aligned} \tag{2}$$

Inductive Logic Programming

Idea: Based on the background knowledge $\mathcal{B}$, to derive a set of rules $\mathcal{R}$ for a target predicate such that it [Muggleton, 1991]:

- entails all the positive examples $\mathcal{P}$
- does not entail the negative examples $\mathcal{N}$

Inductive Logic Programming-Example

Example - Even numbers [Evans and Grefenstette, 2018], learn target predicate $even(X)$:

- $\mathcal{B} = \{zero(0), succ(0,1), succ(1,2), succ(2,3), succ(3,4), succ(4,5)\}$
- $\mathcal{P} = \{even(0), even(2), even(4)\}$
- $\mathcal{N} = \{even(1), even(3), even(5)\}$

Possible Solution

$$\begin{aligned} even(X) &\leftarrow zero(X) \\ even(X) &\leftarrow even(Y), succ2(Y,X) \\ succ2(X,Y) &\leftarrow succ(X,Z), succ(Z,Y) \end{aligned} \tag{3}$$

∂ILP

∂ILP

∂ILP:

- Combines advantages of ILP and the neural-network-based systems - Neurosymbolic (NeSy) AI.
- Has two types of predicates:
  - The intensional P_i, target and auxiliary predicates $\{p_a\}$,
  - Yhe extensional P_e, from the background knowledge ($zero(X)$, $edge(X, Y)$, ...)
- Generates a list of possible definite clauses for intentional predicates based on the program template [Tausend, 1994]

∂ILP - PROGRAM TEMPLATE

Each intensional predicate p_a is defined by its arity, and is described by two clauses according to rule templates $(\tau^1_{p_a}, \tau^2_{p_a})$.

$$\tau = (n_\exists, int) \tag{4}$$

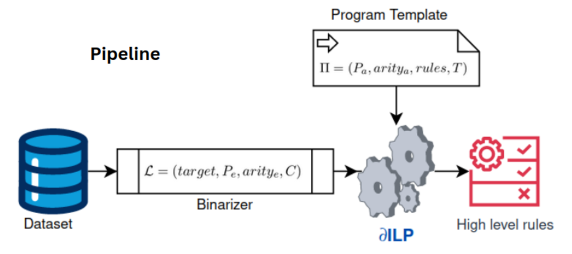
- $n_\exists$ - the number of existentially quantified variables allowed in the clause
- int is a flag that determines whether the atoms in the clause can use intensional predicates P_i

∂ILP learns a set of clauses to minimize the loss function.

Fraud Detection Framework

Pipeline

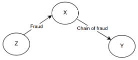
- *Dataset* - Paysim [Lopez-Rojas et al., 2016], and synthetically generated scenarios
- *Binarizer* converts numerical columns to binary and creates sets of facts, positive and negative examples.
- *High level rules* generated SQL query from the derived rules



RESULTS

Chain of fraud

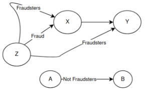
- Transaction(X,Y) - X sends transaction to Y
- Fraud(Z,X) - Transaction from Z to X is Fraud
- Fraud_Chain(X,Y) - X and Y are in a chain of a fraud event
- Template $\tau^1_{target} = (n_\exists = 1, int = 0)$
- Rule $Fraud\_Chain(X,Y) \leftarrow Fraud(Z,X), Transaction(X,Y)$



	orig	destination	Fraud_chain--orig--destination
0	16058	16066	False
1	16065	16052	False
2	16036	16067	False
3	16086	16014	True
4	16043	16004	False
5	16011	16067	True
6	16002	16011	False
7	16051	16086	False
8	16080	16077	False

Fraud Relationship

- Fraudsters(X,Y) - X and Y are Fraudsters
- Fraud(X,Y) - Transaction from X to Y is Fraud
- Template $\tau^1_{target} = (n_\exists = 0, int = 1)$ $\tau^2_{target} = (n_\exists = 1, int = 1)$
- Rule $Fraudsters(X,Y) \leftarrow Fraud(X,Y)$ $Fraudsters(X,Y) \leftarrow Fraud(Z,Y), Fraudsters(Z,X)$



	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

PaySim Dataset

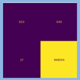
- Binarised Columns from Decision Trees Thresholds
- X - transaction id
- Template

$$\tau^1_{target} = (n_\exists = 0, int = 1)$$
$$\tau^2_{target} = (n_\exists = 0, int = 1)$$
$$\tau^1_{p1} = (n_\exists = 0, int = 1)$$
$$\tau^2_{p1} = None$$
$$\tau^1_{p2} = (n_\exists = 0, int = 0)$$
$$\tau^2_{p2} = None$$

- Rule

$$isFraud(X_0) \leftarrow NOT\{oldbalanceDest > -0.007\}(X_0), pred2(X_0)$$
$$isFraud(X_0) \leftarrow pred2(X_0), amount > 1.297(X_0)$$
$$pred1(X_0) \leftarrow NOT\{oldbalanceDest > -0.007\}(X_0), pred2(X_0)$$
$$pred2(X_0) \leftarrow external\_dest(X_0), type\_TRANSFER(X_0)$$

Performance	∂ILP	DT	DSC
Accuracy	0.999	0.999	0.999
Precision	0.973	0.971	**0.984**
Recall	0.501	0.665	0.501
F1	0.662	0.789	0.664
MCC	0.698	**0.803**	0.702



Summary

Pros/Cons

- Pros:
  - ∂ILP generalizes from a small amount of data - Paysim results based on 1000 from the database out of 6 million transactions
  - Can create recursion predicates
  - Provided shorter explainable rules than the Decision Tree approach

PROS/CONS

- Cons:
  - Scalability
  - Did not outperform other techniques - showed low recall
  - Expert knowledge - Program Template definition

Future directions

- One of the possible future work directions is to apply another neurosymbolic extension for ILP to the scenarios covered in this thesis.

References I

Evans, R. and Grefenstette, E. (2018).
Learning explanatory rules from noisy data.
Journal of Artificial Intelligence Research, 61:1–64.

Lopez-Rojas, E., Elmir, A., and Axelsson, S. (2016).
Paysim: A financial mobile money simulator for fraud detection.
In *28th European Modeling and Simulation Symposium, EMSS, Larnaca*, pages 249–255. Dime University of Genoa.

Muggleton, S. (1991).
Inductive logic programming.
New generation computing, 8:295–318.

Tausend, B. (1994).
Representing biases for inductive logic programming.
In *European Conference on Machine Learning*, pages 427–430. Springer.