UNIVERSITÀ DEGLI STUDI DI MILANO

FACOLTÀ DI SCIENZE POLITICHE, ECONOMICHE E SOCIALI

Master of Science in Data Science for Economics

TRANSACTION PREDICTION ON FINANCIAL NETWORKS

Supervisor: Prof. Sabrina Tiziana GAITO
Co-supervisors: Dott. Matteo ZIGNANI

Thesis by:
Matteo Biglioli
Student ID: 938199

Academic Year 2023-2024

Aknowledgments

First of all, I wanted to thank Manuel, Cheick, and Matteo for the help that they gave me in these last months, your contributions and support were fundamental to the development of this thesis.

And now it's time to speak about you, if you are reading this you are probably one of the people that I had the honor to have around in these last years. I might take some time to thank all of you, but, as you contributed to making me the person that I am now, I think you deserve at least a proper recognition for it.

Let me start by expressing my deepest gratitude to my family; even if most of the times I take you for granted I now realize how relevant your support and your patience were in these past years. While I usually don't involve you directly in my "professional" life, you are always around providing the right amount of guidance for me to keep going.

Speaking of people who are always pushing me I can't help but mention my closest group of friends. Thank you, Navo, Nadir, Fede, and Ale for taking good care of me for basically my whole life. It's not easy to have someone as distracted and emotional as me around, but you accepted every aspect of me and you never stopped helping me grow, even when it meant arguing and repeating things millions of times.

Another warm recognition goes to Mic, Erica, Vale, and Fra. Thank you for making everything fun and easy, for accepting my forgetfulness and for always being there when I needed it. I would never have expected to find such remarkable people just by sharing music tastes on online groups.

Another big thank you goes to you Cri, you have been by my side for most of my life as of now. You contributed to shape the person that I am now and for that, I cannot be grateful enough. I hope one day you will realize how inspired I am by

your strength and dedication.

Now it's time to remember the friends that I made during this (not so short) journey. Thank you, Eli, Robi, Ruben, Martin, Gas, Elisa, Luca, and Lore for sharing this experience with me. I think that I would not have pursued this master's like I did if it weren't for you all. Working with you made this course something more than just a bunch of lessons.

Now it's the time to acknowledge the person who taught me everything I know in my professional life, Matte. I still remember the day when I looked at your GitHub profile and I thought "Well, I need to work with him"; and it was one of the best decisions of my whole life. You were not only a boss, but an amazing mentor and a great friend.

I also wanted to thank you, Maria. While you appeared quite recently in my life you already managed to turn it upside down and now, because of you, I'm writing this thing from a random Barcelona office. You help me balance my negativity with a fire that I never found in anyone.

Speaking about Barcelona there are three persons that I wanted to put here, but I don't think that a few words could even reach the level of gratitude that I want to express.
Thank you Alessia for welcoming me into your life, your energy is something that I've been missing without realizing it.
Thank you, Sara, for having the patience of pushing me every time that I needed it and for sharing this new adventure with me, you make me feel at home.
Thank you Sonia for, I don't know, like, everything. Seeing the world through your eyes is something that everyone should be able to experience. As I told you many times, you may not realize it but you changed my life.

And of course thank you Barcelona crew, Manu, and Carlos. You accepted me into your group without thinking twice about it, I will always be grateful for that. Gràcies Nat per donar-me un cop de mà en aquests mesos de novetats i confusió.

The last words here are for you Mario, I think this is the first time someone mentions you in a long time. Thank you, I made it here.

Contents

Abstract

Financial networks are complex systems characterized by intricate interactions among various entities, such as banks, investors, and markets. Analyzing the dynamics of these networks is crucial for understanding their health and evolution.
Link prediction, a technique aimed at forecasting potential connections between entities, serves as a valuable tool in this endeavor.
This thesis explores the application of link prediction in financial networks, with the goal of developing an innovative benchmarking framework based on the Steemit dataset, a decentralized social media platform leveraging the Steem blockchain to power its operations.

We analyze how standard Machine Learning models behave in this task, fine-tuning them and varying the depth of historical information they have access to. We then compare the results obtained with a simple memorization-based baseline, named EdgeBank, proposed as a *a strong and necessary baseline for future methods to compare against* [1]. In this way, we construct a benchmark in this novel setting which takes into account also the temporal dimension.
Furthermore, to understand how our models' results position themselves in the state of the art of this setting we experiment with the ROLAND [2] framework, in particular with a model called T3GNN[3].

The results of our experiments show that, while feature-engineered models are mostly overlooked in recent literature, they still demonstrate performances that are comparable with more sophisticated models like Graph Neural Networks.
We also highlight that these performances are not easy to reach, because, while GNNs work by generating embeddings directly over the raw data, their competitors require a step of feature engineering which is not always feasible depending on the size of the dataset at hand.

Chapter 1

Introduction

Financial systems are composed of intricate webs of interactions among various entities such as banks, investors, and markets. Understanding the dynamics of financial networks is of extreme importance for policymakers, investors, and researchers, as it enables informed decision-making, risk mitigation, and the formulation of effective regulatory frameworks.

To properly analyze these kinds of networks, we need to be able to predict future connections or links between entities. Link prediction is a network science technique that offers powerful tools to forecast potential interactions within these networks. By leveraging historical transaction data and network topology, link prediction methods give valuable insights into the evolving structure and behavior of financial systems.

This thesis explores the application of link prediction in financial networks, to develop an innovative benchmarking framework. The framework is based on the Steemit dataset, a decentralized social media platform leveraging the **Steem** blockchain to power its operations. The project was launched in 2016 by Ned Scott, and it has quickly become a popular platform for users to share content, engage with others, and earn rewards.

The first goal of our research is to examine the behavior of standard machine learning models, among them KNN, Random Forest, and MultiLayer Perceptron when leveraged for this kind of task. We start by understanding how varying the historical depth of information available impacts the evaluation metrics of link prediction algorithms. We then introduce a baseline model, named EdgeBank, which employs memorization-based techniques. This baseline serves as a benchmark against

more sophisticated algorithms, providing valuable insights into the relative performance of different approaches.

Moreover, to evaluate our findings within the broader landscape of link prediction in financial networks, we experiment with the ROLAND framework [2], an innovative framework that allows us to apply Graph Neural Networks to Temporal Graphs, specifically focusing on the T3GNN model. In this way, we aim both to benchmark this model and to understand the goodness of our first standard machine learning approach against more advanced techniques like the ones reported in the T3GNN paper[3].

The results of our experiments point towards the fact that feature-engineered models, often overshadowed by more advanced models like Graph Neural Networks (GNNs) in literature, still hold considerable performances in the field of dynamic link prediction. Despite the complexity and flexibility offered by GNNs in capturing intricate network structures, our findings reveal that traditional feature-engineered models can achieve performances that are not only competitive but sometimes even comparable.
Unlike GNNs, which operate by directly generating embeddings from raw data, their feature-engineered counterparts necessitate a preliminary step of conceiving and computing relevant features. This process can be extremely expensive and not only requires domain expertise to identify and extract informative features, but its feasibility can be contingent upon the size of the dataset at hand.

In summary, this thesis aims to advance our understanding of financial networks with the application of link prediction techniques. By developing a complete benchmark and conducting different empirical analyses, we wanted to provide valuable insights to people studying and working in the field. Our main finding is related to the in-depth comparison between standard ML techniques, requiring a first step of feature engineering, and more advanced Graph Neural Networks which can be applied in a Temporal Graph setting due to the findings reported in [2].

The following chapters are structured as follows.

Chapter 2 lays down all the theoretical concepts and state-of-the-art findings; more specifically, there are two sections composing this chapter:

- Section 2.1 describes what is a Temporal Graph and provides a formal definition which we will use in the following chapters;

- Section 2.2 lists state-of-the-art methods and models in the field of Dynamic Link Prediction;

Chapter 3 offers a bird's eye view of Blockchain-based Online Social Networks (**BOSNs**) and investigates the Steemit dataset.
Chapter 4 provides a first introduction to the task we are going to solve and on the dataset construction setting, explaining in detail both the feature engineering step and the methodology implemented to work with a Temporal Graph.
Chapter 5 shows all the results achieved with the methods and models introduced in the previous chapter, focusing on exploring the different training settings employed.
Chapter 6 discusses the results and provides ideas for future work to further research on different frameworks that would expand the current knowledge on the field.

Chapter 2

Background and related works

2.1 Temporal Graphs

In this section, we provide a formal definition of Temporal Graph and we analyze the different ways in which it could be represented.

2.1.1 Formal Definitions

To define what is a temporal graph we must start by introducing the concept of graph, since, informally speaking, a temporal graph is basically a graph that changes over time.

Definition 1. *A **Static Graph** is a tuple $G = (V, E, X^V, X^E)$, where V is the set of nodes, $E \subseteq V \times V$ is the set of edges, and X^V , X^E are d_V-dimensional node features and d_E-dimensional edge features.*

Now that we have a formal definition of a **Static Graph** we can introduce the following:

Definition 2. *A **Temporal Graph** is a tuple $G_T = (V, E, V_T, E_T)$, where V and E are, respectively, the set of all possible nodes and edges appearing in a graph at any time, while*

$$V_T := \{(v, x^v, t_s, t_e) : v \in V, x^v \in \mathbb{R}^{d_V}, t_s < t_e\}$$
$$E_T := \{(e, x^e, t_s, t_e) : e \in E, x^e \in \mathbb{R}^{d_E}, t_s < t_e\}$$

are the temporal nodes and edges, with time-dependent features and initial and final timestamps.

In other words, a **Temporal Graph** is defined as a mathematical object that can be seen as a **Static Graph** given a fixed time t but varies as time moves.
We observe that, in this definition, we are assuming that the existence of an edge $e_t := (v, u) \in E$ at any point t in time, implies the existence of the two nodes $v, u \in V$ on which the edge is built on. We further observe that there is no assumption on the domain of t_s, t_e ; this would allow us to work with both **Discrete Time Temporal Graph**, in which $|T| = |\mathbb{N}|$ and **Continuous Time Temporal Graph**, in which $|T| = |\mathbb{R}|$.
In the case of this project, we will be working in a discrete-time setting.

2.1.2 Representations

We now introduce the two main categories for the representation of time-varying graphs.

We start by presenting the **Event-Based** strategy, which focuses on taking into consideration all the insertion and deletion events that happen during the graph lifespan (assuming they are consistent with each other).

Definition 3. *Let G_T be a Temporal Graph, and let ϵ denote one of the following events:*

- ***Node insertion*** $\epsilon_V^+ := (v, t)$*: the node v is added to G_T at time t, i.e., there exists $(v, x^v, t_s, t_e) \in V_T$ with $t_s = t$.*
- ***Node deletion*** $\epsilon_V^- := (v, t)$*: the node v is removed from G_T at time t, i.e., there exists $(v, x^v, t_s, t_e) \in V_T$ with $t_e = t$.*
- ***Edge insertion*** $\epsilon_E^+ := (e, t)$*: the edge e is added to G_T at time t, i.e., there exists $(e, x^e, t_s, t_e) \in E_T$ with $t_s = t$.*
- ***Edge deletion*** $\epsilon_E^- := (e, t)$*: the edge e is removed from G_T at time t, i.e., there exists $(e, x^e, t_s, t_e) \in E_T$ with $t_e = t$.*

An Event-based representation of a Temporal Graph is a sequence of events

$$G_T^E := \{\epsilon : \epsilon \in \{\epsilon_V^+, \epsilon_V^-, \epsilon_E^+, \epsilon_E^-\}\}$$

Another useful representation technique is called **Snapshot-Based**, which focuses on viewing the temporal graph as a sequence of static graphs:

Definition 4. *Let $t_1 < t_2 < \ < t_n$ be the ordered set of all timestamps t_s, t_e occurring in a Temporal Graph G_T . Set*

$$V_i := \{(v, x^v) : (v, x^v, t_s, t_e) \in V_T, t_s \leq t_i \leq t_e\}$$

$$E_i := \{(e, x^e) : (e, x^e, t_s, t_e) \in E_T, t_s \leq t_i \leq t_e\}$$

and define the snapshots $G_i := (V_i, E_i), i = 1, ..., n$.
Then a Snapshot-based Temporal Graph representation of G_T is the sequence

$$G_T^S := \{(G_T, t_i) : i = 1, ..., n\}$$

of time-stamped static graphs.

Before moving on to the next section, we draft a small comparison between the two representations for the reader to understand why we will choose the **Snapshot-Based** one.

While both representations are equally viable, current representation learning algorithms have been mostly designed for the **snapshot-based** one, with only a few works on learning from **continuous-time dynamic graphs**[4]. We can assume two underlying reasons for this:

1. The snapshot-based representation could be easier to understand and to work with, as we can work with models just by showing them a different graph each time and asking them to predict the next snapshot. In this way, we do not need to work with multiple types of events that would need different ways to handle them.

2. For most of the projects in which Dynamic Link Prediction is performed (and even more in financial-based ones), it is not needed to have an extremely granular approach as provided by the continuous-time representation. This allows us to work with snapshot-based graphs in which the time window between two snapshots can be longer (in our case 2 weeks).

2.2 Dynamic Link Prediction

In this section, we introduce and formalize the main task that we will be trying to approach in this project, which is **Dynamic Link Prediction**.

We define the task as follows [5]:

Definition 5. *Let $G_T = (V, E, V_T, E_T)$ be a TG. The temporal link prediction task consists of learning a function*

$$f_{LP} : V \times V \times R^+ \longrightarrow [0, 1]$$

which predicts the probability that, at a certain time, there exists an edge between two given nodes.

Basically saying that our goal would be for the estimator to predict the existence of a link between two given nodes, by taking as input the tuple of nodes and a specific time.

While at first glance this task may seem quite straightforward, we must highlight that there is a subtle difference regarding how it can be performed based on the learning setting considered.
In machine learning, we distinguish between:

- **Inductive Learning**: A model is trained on a subset the dataset (*training set*) and then applied to the remaining data (*test set*).
- **Transductive Learning**: A model is provided with the whole dataset of which only a part is already labeled (*training set*) and its goal is to infer the value of unlabeled data points.

This distinction can be seen in the Dynamic Link Prediction task, in particular, we define the following:

Definition 6. *Assume that a model is trained on a set of $n \geq 1$ temporal graphs $G_{\mathcal{T}} := \{G_T^i := (V_i, E_i, X_i^V, X_i^E), i = 1, ..., n\}$. Moreover, let*

$$T_e^{all} := \max_{i=1,...,n} T_e(G_t^i), \quad V^{all} := \bigcup_{i=1}^{n} V_i, \quad E^{all} := \bigcup_{i=1}^{n} E_i$$

be the final timestamp and the set of all nodes and edges in the training set. Then, we have the following settings:

- ***Transductive learning****: inference can only be performed on* $v \in V^{all}, e \in E^{all}$*, or* $GT \subseteq_V G_T^i$ *with* $G_T^i \in G_{\mathcal{T}}$*.*
- ***Inductive learning****: inference can be performed also on* $v \notin V^{all}, e \notin E^{all}$*, or* $G_T \not\subseteq_V G_T^i$ *, for all* $i = 1, ..., n$*.*
- ***Missing link prediction****: inference is performed for* $t \leq T^{all}$*.*
- ***Future link prediction****: inference is performed for* $t > T^{all}$*.*

In this project, we will focus on predicting **Future** links in an **Transductive Learning** setting.

Chapter 3

Dataset

3.1 Blockchain-based Online Social Networks

In the past years, the growth of blockchain technology has fostered multiple small revolutions in different sectors, including the one of online social networks. Blockchain-based online social networks represent a new era for social networks, offering a decentralized infrastructure that fundamentally alters the dynamics of data ownership, privacy, and security.
This decentralized architecture eliminates the need for a central authority, such as a social media platform, to control user data and govern interactions.

One of the key advantages of blockchain-based social networks is the ability to empower users with greater control over their data.
This shift towards user-centric data ownership not only enhances privacy but also helps the system to be more equitable.

However, the integration of blockchain technology into online social networks also presents different challenges:

- Scalability is a huge issue, as the consensus mechanisms and data storage requirements of blockchain networks are not easy to handle in a decentralized architecture.
- Regulatory compliance is still an obstacle to the mainstream adoption of blockchain-based social networks.
- User experience issues, such as the complexity of cryptographic key management, are still not completely solved and are blocking the adoption.

3.2 Case study: Steemit

The dataset on which this discussion is based comes from **Steemit**, a decentralized social media platform leveraging the **Steem** blockchain to power its operations. The project was launched in 2016 by Ned Scott, and it has quickly become a popular platform for users to share content, engage with others, and earn rewards.

The Steem blockchain is used to store a variety of data, including user profiles, content, and voting records. It is also the foundation for STEEM and SBD tokens, which are used to reward users for their contributions to the platform. Steemit uses a token-based economy, which means that users can earn rewards for creating and curating content.

In a recent study, Ba *et al.* [6] showed that economic/financial factors, such as the price of the STEEM cryptocurrency, may influence the social network and the financial interactions supported by Steemit. In the paper, we can find a complete analysis of the price trends of STEEM and the correlations that this factor has over the network evolution.

3.3 Dataset Description

The dataset is a sequence of snapshot graphs collecting all the transactions that took place on the blockchain during 2 weeks. In particular:

- **Snapshot (0)** is a graph containing all the transactions between users that happened between 2016-03-24 (the starting date of the project) and 2016-12-31.
- **Snapshots (1, ..., 25)** are snapshot graphs, each one containing transactions of two weeks for the whole of 2017. (e.g. snapshot 1 contains transactions that happened between 2017-01-01 and 2017-01-15)

We highlight that, while the complete dataset also contains textual data (posts and comments of users), for the whole scope of the project we will only rely on the architectural data of the graph.

We will now present a more detailed description of the dataset, focusing more on its technical features as a graph rather than its context.

The Temporal Graph contains a total of 14814 unique nodes and 75731 unique edges, even if, as expected most of them are present only in snapshot 0, which collects data related to the 2016 transactions. In Figure 1 we show the time distribution of nodes and edges (without taking into account snapshot 0 which was out of scale). The only peculiarity noticeable here is the strong increase both in number of nodes and edges at snapshot 13, which can be strongly correlated to a period of drop in performance in July 2017, following a rise in price a few months prior, in May 2017[6]. In Figure 2 we report the time distribution of the density, which follows the same explanation given above related to the number of nodes and edges.

Because, as said above, in this project we focus on predicting **future** links in an **inductive learning** setting, we define a time t_{split} which split the timeline into two subsets: E_{train} and E_{test}. Our models will be able to train only on edges for which $t < t_{split}$ and will need to predict edges for which $t > t_{split}$.
This splitting divides edges into three main sets:

1. Edges which are seen only in training.
2. Edges which are seen both in training and test (*transductive edges*).
3. Edges which are seen only in test (*inductive edges*).

Based on this distinction we can speculate that inductive edges will be harder to classify than transductive edges, that is because the former will come to the models as a "surprise", given that they were never seen during the training phase.

To synthesize these intuitions in a formal setting we introduce two important indices[1] which can help us to better grasp the effort that our prediction models have to apply to classify a new possible link as present or not, defined below:

Definition 7. *We define* ***Reoccurence Index*** *as the ratio between the number of transductive edges and the overall number of edges in the training set.*

$$\frac{|E_{train} \cap E_{test}|}{|E_{train}|}$$

Definition 8. *We define* ***Surprise Index*** *as the ratio between the number of inductive edges and the overall number of edges in the test set.*

$$\frac{|E_{test} \setminus E_{train}|}{|E_{test}|}$$

We reported the two indices in table A in appendix A.

Finally, we show in Figure 3 the **Temporal Edge Appearence** (TEA) plot, which shows the portion of repeated edges versus newly observed edges for each timestamp [1].
From the plot we can see that at timestamp 14 the number of newly observed edges increased, while the number of repeated ones decreased; this behavior is in line with the explanation given above regarding the spike in the number of transactions. What could have happened here is that consequently of the rise in price that happened in May, people who were already enrolled in the platform increased their usage exponentially; that is reflected in the spike that we see both in Figure 1 and 3 which show us an increase in the number of transactions that already happened in the past (repeated edges). Following that the network experienced a drop in performance, which led to a decrease in the number of transactions which is even more noticeable when we look directly at repeated edges.

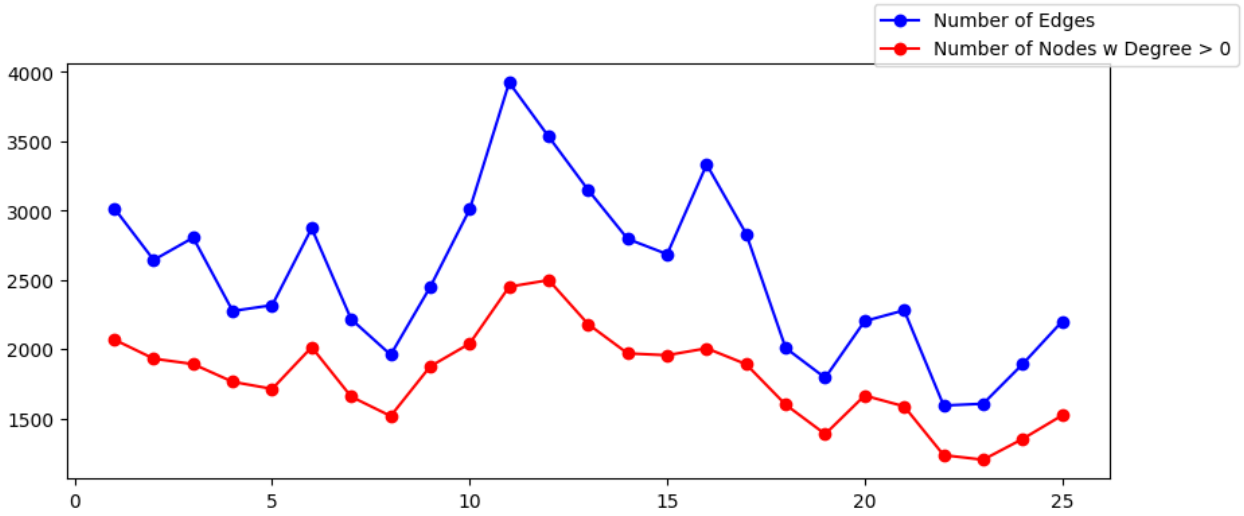

Figure 1: Number of Edges and Nodes with Degree greater than 0

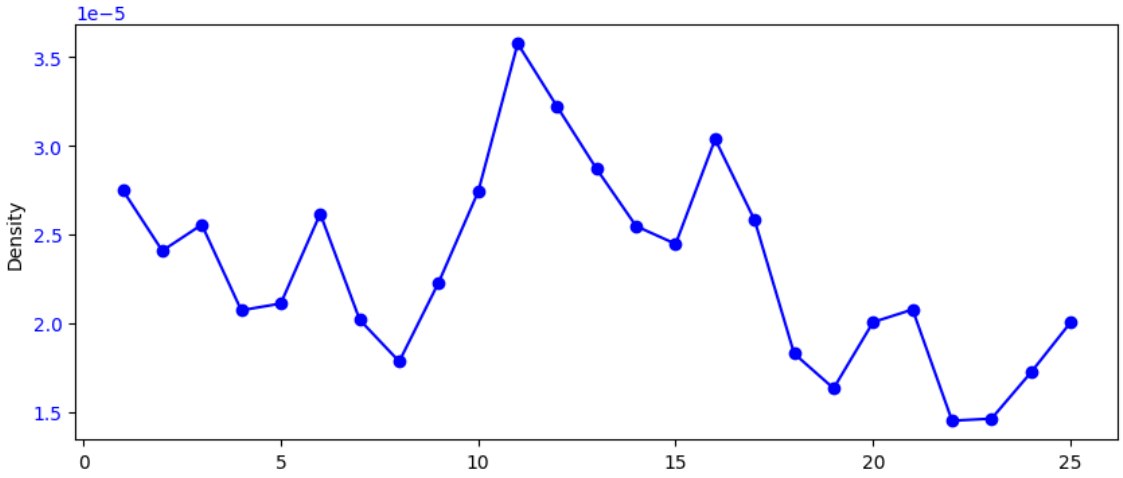

Figure 2: Density time distribution

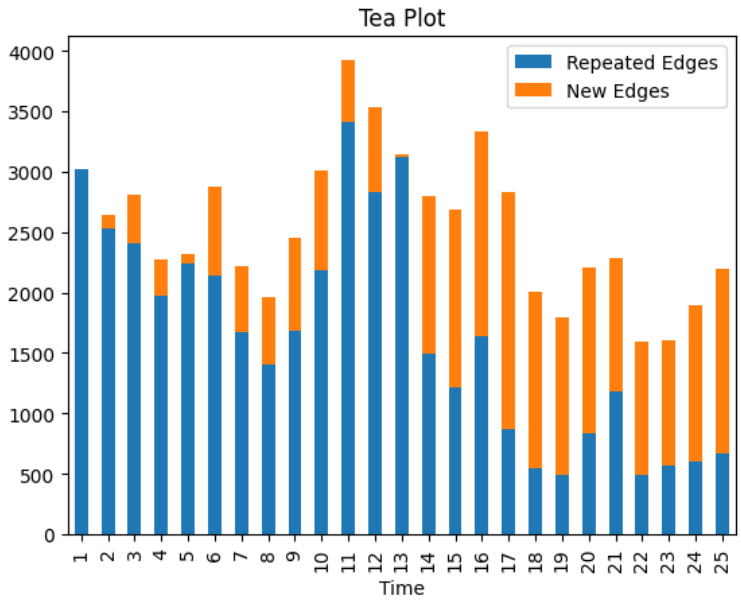

Figure 3: TEA Plot

Chapter 4

Methodology

Our goal here would be to compare different state-of-the-art Machine Learning models with T3GNN[3], but, to perform this task given the obvious computational constraint, we decided to take a conservative approach which allowed us to fine-tune different predictors in a controlled setting before experimenting with them in what we will call the live update setting.

Before diving into the topic we will briefly explain the two settings:

- **Fixed Split**: this setting reflects what is usually done in Machine Learning projects, where the whole dataset is partitioned in two sets (training and test) given a fixed threshold, in our case 80%. Then the model is trained on the first set and evaluated on the second one.
- **Live Update**: in this scenario, we do not find a clear fixed split between training and test sets. We start by training the model on the first snapshot and storing its prediction for the second one; we then proceed to *re-train* it on the second snapshot and ask for the prediction for the third etc. Basically at step n the model will have seen all data related to the snapshots $1, ..., n$ and it will be asked to predict snapshot $n+1$. Finally, we evaluate all the predictions made and we average the results.

We will start by evaluating different predictor families on a Fixed Split Setting, to exclude the ones that perform worst and to find the best hyperparameters for the ones that perform best. We will then re-evaluate only the best models in a Live Update Setting in which we will be able to make a comparison between them and T3GNN.
Note that we will also compare with the EdgeBank baseline in all of these steps.

4.1 Data Modeling

4.1.1 Negative Sampling

Before diving into the actual prediction models we need to present the method which we used to perform negative sampling, i.e. to extract, both for training and test data, a sample of negatively labeled edges.

First of all, we understand that the sampling process is necessary because the number of possible nodes of a directed graph is $n * (n - 1)$, where $n = |V|$ is the number of nodes. In our case, we would have had ~ 220 Million edges, of which just a fraction (~ 2000) are positive. This would have led to multiple issues, the most important being the fact that the dataset would have been strongly unbalanced and that a commercial PC would have had serious performance issues in evaluating a model hundreds of millions of times.

Now that we are aware of the issue at hand and of the constraint that we have to respect to solve it, we present three different strategies[1] that could be applied:

1. **Random Sampling**: This is the easiest among all strategies, and it consists of generating negative edges by keeping the same source nodes of positive edges but changing the destination. While being extremely fast and easy to implement, this method usually does not implement **Collision Checking**, meaning that it is at risk of randomly sampling a positive edge among the negative ones. Most importantly it does not "challenge" the prediction model at all because, due to the sparsity of the graph, it is extremely easy to sample negative edges both in training and test sets, thus training a model which could be outperformed by a simple memorization-based baseline (**EdgeBank**).

2. **Historical Negative Sampling**: In this strategy, we extract edges from the set of edges that have been observed in the past but are not present in the current snapshot. In this way, we are inherently checking for collisions and challenging our models by providing more hard-to-classify edges.

3. **Inductive Negative Sampling**: In this strategy, we extract edges from the set of edges that have not been observed in the past, but they are in the test set in a snapshot different than the one we are predicting now ($e \in E_{test} \cap \overline{E_{train}} \cap \overline{E_t}$). In this way we are both checking for collision and challenging our models even more because we are providing edges that were never seen before

but we are sure they will be seen sometime, so we expect them to be hard to classify.

In this project we mainly focus on **Historical Negative Sampling** because it is the strategy used in the paper presenting T3GNN [3] which is the state-of-the-art model.

4.1.2 Dataset Generation

This section is dedicated to presenting the practical steps to generate the dataset which we will use to evaluate the different models. We show the process that we followed to work with Temporal Graphs and we report the strategy we leveraged to address the negative sampling problem.

First of all, we need to compute the different features on the Temporal Graph leveraging the negative sampling strategy reported in section 4.1.1. To do so we performed the following steps:

1. We split our Temporal graph into multiple snapshots.
2. For each snapshot at $\bar{t} > 0$ we extract the subset of positive and negative edges by leveraging the **Historical Negative Sampling** strategy.
3. For each snapshot we compute the four coefficients reported above considering only the couple of nodes that are forming the edges sampled at step 2.
4. We save the resulting dataset into multiple Pickle files for performance reasons.

Note that this alone cannot be considered a ready-to-go dataset, because, for each edge $e = (u, v)$ sampled at time $\bar{t}$, we computed its features only at time $\bar{t}$ which cannot be used to predict the existence of an edge.
This is because if we wanted to predict the existence of an edge at time $\bar{t}$, we should only consider features that can be computed at time $t < \bar{t}$.

To do so we need to perform another important step which is to consider, for each edge $e = (u, v)$, the past features of its nodes u, v computed at times $t < \bar{t}$.
In particular, we assume a fixed value k which is the number of past snapshots that we want to consider to collect features and we perform the following:

1. We again split our Temporal graph into multiple snapshots.

2. For each snapshot at $\bar{t} > 0$ we extract the subset of positive and negative edges by leveraging the **Historical Negative Sampling** strategy.

3. For each edge we compute the four coefficients reported above on its source/-target nodes but only on the snapshots at time $\bar{t} - 1, ..., \bar{t} - k$.

4. We save the resulting dataset into multiple Pickle files for performance reasons.

In our project we let k vary between 1 and 5, both for performance reasons and because we noticed that all our evaluation metrics reached a plateau at $k = 4$. This would mean that to accurately predict the existence of a link, our models just need to know the behavior of its source/target nodes in the previous 4 snapshots, any more information would be redundant or useless.

Finally, as it is commonly done in projects of this kind we simply split the dataset into training and test, keeping 80% of the data for the training. Note that this split is performed "snapshot-wise", meaning that given 25 snapshots we keep the first $25 \times 0.8 = 20$ snapshots in the training set.

Having built the dataset as explained above, the procedure becomes quite straightforward, following the best practices of a common Machine Learning project.

4.2 Link Prediction Models

As explained in section 2.2, we will try to train different models to predict the probability that, at a certain time, there exists an edge between two nodes; specifically focusing on predicting Future links in an Inductive Learn- ing setting.

To do that we leverage two approaches, explained in the following sections.

4.2.1 Feature-Engineered Based

As reported at the beginning of this chapter, in this project we leveraged only structural features of the graph, without taking into account additional pieces of information stored in the Steem blockchain such as user-generated content. This was done on purpose because the goal of the project is to compare basic Machine Learning models against a more complex GNN which can be leveraged in a Temporal Graph

setting thanks to the findings of the ROLAND framework [2].

While the code is written to be able to subselect different structural features, we always work with the four main indexes provided by NetworkX [7] which are the following:

- **Jaccard Coefficent**: A similarity measure used to compare two sets of data. It is defined as the size of the intersection of the two sets divided by the size of the union of the two sets. The Jaccard coefficient can take on values between 0 and 1. A value of 1 indicates that the two sets are identical, while a value of 0 indicates that the two sets have no overlap. In our case, the coefficient is computed over the neighborhood of any two nodes as follows:

$$\frac{|N(u) \cap N(v)|}{|N(u) \cup N(v)|}$$

  where $N(u)$ denotes the neighborhood of node u

- **Adamic Adar Index**[8]: A similarity measure used to compare nodes in a graph. It is a weighted version of the Jaccard coefficient that takes into account the degree of each node. The Adamic-Adar Index can take on values between 0 and 1. A value of 1 indicates that the two nodes have a very strong connection, while a value of 0 indicates that the two nodes have no connection.

$$\sum_{\omega \in N(u) \cap N(v)} \frac{1}{log|N(\omega)|}$$

- **Resource Allocation Index**: Another similarity measure used to compare nodes in a graph which is again a weighted version of the Jaccard coefficient, this time taking into account the resource allocation of each node. The resource allocation of a node is a measure of the node's importance in the graph. The RAI can take on values between 0 and 1. A value of 1 indicates that the two nodes have a very strong connection, while a value of 0 indicates that the two nodes have no connection.

$$\sum_{\omega \in N(u) \cap N(v)} \frac{1}{|N(\omega)|}$$

- **Preferential Attachment**[9]: A similarity measure that assesses the connection between nodes based on their relative connectivity. It posits that nodes with higher connectivity are more likely to be similar due to shared connections. PA is particularly valuable for analyzing dynamic networks and identifying communities. Its strengths lie in capturing real-world network dynamics and its role in dynamic network analysis and community detection. However, it can be sensitive to statistical noise and may overlook connectivity aspects beyond common neighbors.

$$|N(u)|N(v)|$$

These indexes could be computed over any two nodes u, v of the graph. To build our dataset we defined a fixed list of edges for each snapshot graph (leveraging the negative sampling strategy explained above) and we computed these coefficients above over this list of a couple of nodes.

4.2.2 Deep Graph Neural Networks

In this section, we will summarize and report the most important findings related to the ROLAND framework, recently presented as *an effective graph representation learning framework for real-world dynamic graphs.* [2].

Because ROLAND aims to allow Graph Neural Networks (GNN) to be easily repurposed to dynamic graphs, we must start by giving context and defining what GNNs are.
GNNs are a class of neural network architectures specifically designed to process and analyze graph-structured data; they learn a function $h_v = GNN(v, G; \theta)$, with $v \in V$ and θ being a set of trainable parameters [5]. The way they work resembles the one of Convolutional Neural Networks (CNN), in the sense that both these architectures learn embeddings that are based on the information contained in a single data point and its "neighbors".

In the case of CNN, data points are pixels and their neighbors are other pixels that are spatially close, for GNN each node is a data point and its neighbors are other nodes connected to it by edges.
Formally if we assume that the information of a node v is stored in a feature vector

h_v, we can define the feature vector after k iterations of the GNN as

$$h_v^k = COMBINE^{(k)}(h_v^{k-1}, AGGREGATE^{(k)}(\{h_u^{k-1}, u \in \mathbb{N}[v]\}))$$

The idea of building the ROLAND framework comes from the realization that, while Graph Neural networks have been successful in static graphs-related tasks, they are not easily applicable to dynamic graphs due to multiple limitations both related to model design and training strategy.
As presented above, we can define a static GNN as a stack of L layers, and we can define the node embedding matrix (at l-th layer) as $H^{(l)} = \{h_v^{(l)}\}_{v \in V}$.
ROLAND proposes an *update-module* which updates these embeddings hierarchically and dynamically [2] in which:

$$H_t^{(l)} = UPDATE^{(l)}(H_{t-1}^{(l)}, \tilde{H}_t^{(l)}), \quad \tilde{H}_t^{(l)} = GNN^{(l)}(H_t^{(l-1)})$$

While the one presented above is the main insight presented in the paper, which is to view GNN layers as hierarchical node states, there are two other core features that allowed models built using the ROLAND framework to achieve great improvements over state-of-the-art baselines.

- **Incremental Training**: Instead keeping into memory all the previous snapshots $\{G_1, ..., G_{t-1}\}$ when predicting y_t, the models only works on the latest snapshot G_t and H_{t-1} which is the last embedding.
- **Meta-Training**: The prediction task is formulated as a meta-learning problem for which making predictions in different periods represents a different task. In this way a *meta model* $\text{GNN}^{(meta)}$ can be found as an initialization from which to derive a specialized model for future unseen prediction.

4.3 Evaluation Metrics

All of the models presented below and the baseline will be evaluated on multiple metrics that we discuss in this section. Note that while we will always compute all 6 metrics we will mostly focus on the Average Precision which has become the reference evaluation metric for Dynamic Link Prediction tasks[10].

- **Accuracy**: measures the overall correctness of predictions made by a model, calculated as the ratio of correctly predicted instances over total instances. It is a simple and widely used metric but may be misleading in imbalanced datasets.

- **Precision**: the ratio of true positive predictions over total predicted positives. It emphasizes the accuracy of positive predictions, making it useful when the cost of false positives is high.

$$precision = \frac{TP}{TP + FP}$$

- **Recall**: measures the ability of a model to capture all relevant instances, calculated as the ratio of true positive predictions over total actual positives. It is crucial in scenarios where missing positive instances is a significant concern.

$$recall = \frac{TP}{TP + FN}$$

- **F1 Score**: the harmonic mean of precision and recall, providing a balance between the two metrics. It is particularly useful when there is an uneven class distribution.

$$F1\ score = 2 \times \frac{precision \times recall}{precision + recall}$$

- **Average Precision**: summarizes a precision-recall curve as the weighted mean of precisions achieved at each threshold, with the increase in recall from the previous threshold used as the weight:

$$AP = \sum_n (R_n - R_{n-1}) P_n$$

where P_n and R_n are the precision and recall at the $n-$th threshold [11].

Chapter 5

Results

5.1 Fixed Split Setting

5.1.1 Fixed Dataset and Varying predictor family

As discussed above we begin to evaluate different predictors family using a Fixed Split approach. Note that, as reported in section 4.1.2, we can vary the parameter k which controls how much past information is given to the model for each step.
In this first iteration we will fix the value of $k = 3$ as our goal here is only to discard bad-performing families of predictors and to start to get an idea of how different models behave.

We start by generating our dataset as explained in Section 4.1.2, which we then split into train/test using a threshold of 80% - meaning that 20 over our 25 snapshots are used for training purposes.
Then, for each predictors family, we perform a **Grid Search** with a **3-fold Cross Validation** which allows us to perform an exhaustive search over specified parameter values for an estimator. In particular, we focused on optimizing the **Average Precision** metric discussed in the section above.

In appendix B we report the grid search parameter grids of all the predictors families we experimented with

After selecting the best-performing model, we scored it on the test set leveraging all the evaluation metrics reported above. The results are reported in Table 5.1.1

As reported above we will focus mainly on evaluating the **Average Precision** of models while taking into account other evaluation metrics to get more context about

Metric Name	KNN	SVC	Decision Tree	Random Forest
Accuracy	0.827	0.663	0.844	**0.850**
Precision	0.909	**0.917**	0.907	0.902
Recall	0.731	0.366	0.771	**0.788**
F1	0.810	0.523	0.833	**0.841**
ROC AUC	0.828	0.666	0.845	**0.850**
Average Precision	0.800	0.655	0.815	**0.818**

Metric Name	MLP	CatBoost	LightGBM	EdgeBank
Accuracy	0.847	0.847	0.847	0.648
Precision	0.908	0.907	0.903	0.638
Recall	0.777	0.776	0.781	0.701
F1	0.837	0.837	0.838	0.668
ROC AUC	0.848	0.848	0.848	0.647
Average Precision	**0.818**	0.817	0.816	0.598

Figure 4: Grid Search results for Fixed Split Setting and k=3

the performance of each one.

Thanks to this initial analysis we immediately discarded both **C-Support Vector Classification** - due to being the worst performer, with just a small increase over the EdgeBank Baseline - and **Decision Tree**, which performs almost identically to the Random forest but slows us down due to performance reasons.

5.1.2 Varying Dataset and Varying predictor family

We then proceed to reproduce the same analysis of the section above, but this time letting the value of k vary between $1, ..., 5$. As explained before this allows the models to increase the number of past snapshots they have access to, which, as expected, increases their performance.

We perform this analysis on the same predictors' families and parameters grid that are reported above, with the only exception of excluding both **C-Support Vector Classification** and **Decision Tree** for the reasons already explained.

We report here below only the results table related to the analysis performed with $k = 5$, while the rest of the results can be found in appendix C.

Metric Name	KNN	Random Forest	MLP	CatBoost	LightGBM	EdgeBank
Accuracy	0.826818	0.844318	0.834545	0.840000	**0.848182**	0.682727
Precision	0.893514	0.890076	0.894180	**0.895094**	0.893002	0.672255
Recall	0.745266	0.788548	0.761948	0.773219	**0.793959**	0.723174
F1	0.812684	0.836242	0.822785	0.829705	**0.840573**	0.696785
ROC AUC	0.827491	0.844778	0.835144	0.840551	**0.848629**	0.682394
Average Precision	0.794314	0.808459	0.801318	0.806422	**0.812870**	0.625703

Figure 5: Grid Search results for Fixed Split Setting and k = 5

As we can see all models perform great on the task, with comparable results when we take into account the **Average Precision** score. We also notice that the **EdgeBank Baseline** performs definitely worse than the other predictors, showing that the results obtained are not heavily impacted by a bad choice of negative sampling or a high seasonality component of the graph.

Because in this round of experiments, no model is performing clearly worse than the others as happened before, we decided to keep all of them while moving to the next setting which will be more challenging.

5.2 Live Update Setting

5.2.1 T3GNN

We start this new setting by introducing the **T3GNN** model and performing a Grid Search to find its best parameters for the comparison with the benchmarks.
As reported above, this model is leveraging the *ROLAND* framework to be able to apply a GNN to a temporal graph.
While in the paper[3] were explained different models, also leveraging contextual information of the graph, in this project, we will only work with the one relying solely on the graph structure.

We started by developing a bit of code which allowed us to compare this model with the benchmarks that we reported earlier in the project.
It should be noted at this point, and we will report it again when discussing the results, that the setting in which we will compare the different models is not the best one possible for **T3GNN**. While in the paper[3] the model is fine-tuned at every prediction step (between each k and $k + 1$), in this setting this kind of behavior would not be possible for our benchmark models. To provide a fair comparison we

decided to compare **T3GNN** with the other models without varying anything.

As for the other models, we performed a Grid Search to find the best values for the hyperparameters of the model. We report the parameter grid in Table 1.

HyperParameter	**Grid Search Values**
num_gnn_layers	1, 2, 3
update	gru, mlp, linrnn, avg
hidden_dim	32, 64, 128, 256, 512

Table 1: HyperParameters Grid for T3GNN

The Grid Search process revealed that for this setting and dataset, the best configuration is given by 1 GNN layer of dimension 256 which leverages an MLP-based update strategy.

5.2.2 Benchmark comparison

We now proceed to compare the different models in the **Live Update Setting**, meaning that we will feed them consecutive snapshots asking, each time, for a close-range prediction.
As we did in the previous setting, we start by defining a constant k which will determine the number of past snapshots the model will use to predict the next one. We then take only the best models' configurations for each family selected from the grid search process reported in the subsection above and train them only on the first k snapshots.
Note that the value k is used only for the benchmark models, as T3GNN does update its internal embeddings at each step, inherently remembering every past snapshot.

From now on we will keep asking the models to return a prediction for snapshot $k+1$, we will then store this prediction and fine-tune the models by showing them the correct labels for snapshot $k+1$.
We end up with different evaluation metrics which vary in time that we can average to come up with a complete bird's eye view of the situation.

Here below we report:

1. The graph of the behavior of the **Average Precision** over time when considering both 1 and 5 snapshots as past information in Figures 6 and 8

2. The averages of all the different evaluation metrics over time considering both 1 and 5 snapshots as past information in Tables 5.2.2 and 5.2.2

We show the other results in Appendix D

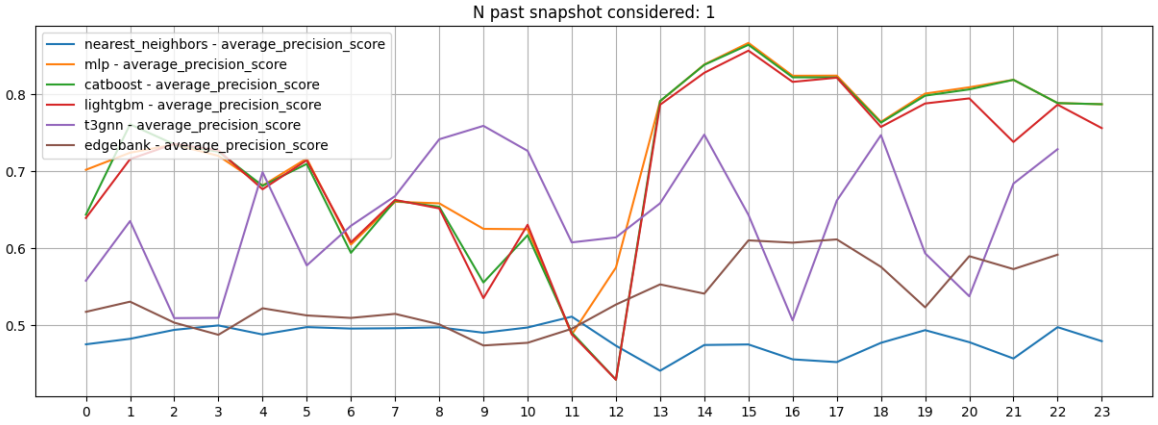

Figure 6: Average Precision Score behavior for 1 past snapshot

Metric Name	KNN	MLP	CatBoost	LightGBM	T3GNN	EdgeBank
Accuracy	0.457224	**0.760177**	0.732347	0.722305	0.598740	0.543190
Precision	0.477582	**0.795260**	0.767885	0.767943	0.704121	0.539940
Recall	**0.889095**	0.717034	0.679304	0.658241	0.527253	0.398847
F1	0.621084	**0.734523**	0.709800	0.696002	0.586991	0.425215
ROC AUC	0.454993	**0.760697**	0.732911	0.722975	0.601857	0.543020
Average Precision	0.482482	**0.726179**	0.714898	0.706004	0.640999	0.536946

Figure 7: Averaged evaluation metrics results for Live Setting and k = 1

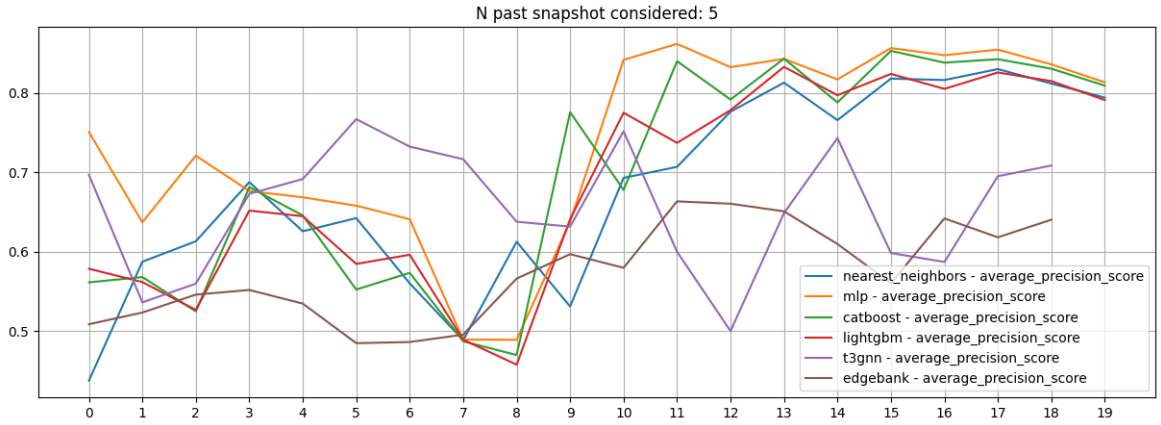

Figure 8: Average Precision Score behavior for 5 past snapshots

Metric Name	KNN	MLP	CatBoost	T3GNN	LightGBM	EdgeBank
Accuracy	0.702655	**0.773322**	0.720444	0.707514	0.604880	0.596936
Precision	0.733818	**0.780076**	0.749773	0.744177	0.719968	0.618060
Recall	0.630865	**0.740378**	0.618662	0.591447	0.548745	0.428022
F1	0.673930	**0.747992**	0.671631	0.654621	0.610574	0.466952
ROC AUC	0.703304	**0.773737**	0.720958	0.708113	0.600471	0.597114
Average Precision	0.680710	**0.738802**	0.697854	0.685806	0.656726	0.574764

Figure 9: Averaged evaluation metrics results for Live Setting and k = 5

The comparison between T3GNN and the feature-engineered models provides interesting insights into the efficacy of these two different approaches. Both in the results shown here and in Appendix D we can see that, while T3GNN, being a neural network-based model, offers powerful capabilities in capturing patterns within the data, the performance of baseline feature-engineered models was surprisingly competitive.

It should be noted here that, to provide a fair comparison, we did not exploit all the possible techniques that would have improved the performance of T3GNN. In particular, we can see that in its presentation[3] this model reached an Average Precision over time of 0.713 on the Steemit dataset, which is comparable to the performance of the best baseline model that we had in the study, the Multilayer Perceptron.

A key result that we can extract from this comparison is that feature-engineered models can perform quite well when compared to deep learning approaches. Even though neural network architectures are more advanced and can be easily fine-tuned to reach even higher performances, the simplicity and interpretability of feature-engineered models make them a feasible alternative, especially in scenarios where the dataset is contained in size, making the computation of features viable as it would not exceed performance limitations.

In conclusion, while neural network models remain the most powerful tool to leverage in these kinds of tasks, both when the dataset size does not allow an easy computation of features and when we want to reach the maximum level of performance, our results show the effectiveness of feature-engineered baselines. The decision to work with a feature-engineering-based approach should be driven by a careful consideration of both computational constraints and specific requirements of the problem at hand.

Chapter 6

Conclusions and future work

In this thesis, we explored the application of different machine learning models to the dynamic link prediction task in financial networks, a binary classification task that aims to predict the evolution of a network over time. Specifically, we focus on developing an innovative benchmarking framework based on the Steemit platform, a novel decentralized Web3 platform that offers cryptocurrency transactions between users with high-resolution temporal information.

First, we model our dataset as a snapshot-based temporal network, allowing us to tweak the past information - the number of snapshots - that is accessible by the models to make predictions. We use real edges as positive examples and we adopted two different strategies to obtain the negative examples.

After having generated the dataset we started experimenting with several feature-engineered based models, discarding the ones which were underperforming compared to the rest. We then proceeded with a grid-search approach to find the best hyper-parameter combination for each one of the remaining models, while still varying the quantity of past information they had access to.

Finally, we compared the best models both with a pure memorization baseline, namely EdgeBank, and T3GNN, a Graph Neural Network model that leverages the ROLAND framework to work in a dynamic setting. The results of this comparison shed light on the dissimilarities between feature-engineered models and GNNs: while performances are comparable, the main difference between the two approaches is represented by the data preparation that needs to be performed.
Because GNNs directly generate embeddings from the graph, no feature engineering

is required; this is extremely useful when we work with huge datasets for which generating features is extremely costly and, at times, even impossible.

Looking ahead, there are a few things that we can propose for future research. Firstly, extending our benchmarking framework to include additional datasets could provide more insights into the robustness of our findings.
Moreover, exploring different feature engineering techniques could help to understand if we already reached the top with feature-engineered models or if there is still space for improvement.
Lastly, we could work with different dataset generation techniques to provide both more robustness to the project and some eventual insights that could help us understand more how the models behave when trained on different datasets.

Appendix A

Reoccurrece and Surprise index

Time	# Nodes w Degree > 0	# Edges	Reoccurrece Index	Surprise Index
0	14809	36812	0.000	0.000
1	2069	3017	0.000	0.000
2	1931	2643	0.032	0.971
3	1892	2804	0.045	0.954
4	1764	2275	0.055	0.938
5	1713	2317	0.052	0.934
6	2014	2872	0.054	0.925
7	1657	2216	0.065	0.901
8	1517	1959	0.052	0.909
9	1875	2448	0.053	0.900
10	2040	3008	0.050	0.897
11	2450	3923	0.042	0.905
12	2498	3533	0.028	0.925
13	2180	3148	0.017	0.942
14	1969	2795	0.001	0.994
15	1956	2685	0.031	0.838
16	2006	3332	0.038	0.778
17	1891	2832	0.044	0.723
18	1599	2008	0.047	0.654
19	1387	1793	0.040	0.653
20	1665	2202	0.038	0.636
21	1587	2280	0.036	0.623
22	1235	1593	0.032	0.600
23	1202	1606	0.030	0.509
24	1352	1892	0.028	0.469
25	1522	2200	0.025	0.400

Table 2: Analysis on the dataset with Reoccurence and Surprise index

Appendix B

Fixed Split Setting Grid Search

HyperParameter	Grid Search Values
n_neighbors	1, 4, 7, 10, 13, 16, 19
weights	uniform, distance

Table 3: HyperParameters Grid for K Nearest Neighbors

HyperParameter	Grid Search Values
C	1, 100, 1000
penalty	l1, l2

Table 4: HyperParameters Grid for C-Support Vector Classification

HyperParameter	Grid Search Values
criterion	gini, entropy, log_loss
max_depth	5, 10, 15, 20, 25, 30, 35, 40, 45, 50

Table 5: HyperParameters Grid for Decision Tree

HyperParameter	Grid Search Values
n_estimators	100, 200, 300, 400, 500, 600, 700, 800, 900, 1000
max_depth	1, 3, 5, 7, 9

Table 6: HyperParameters Grid for Random Forest

HyperParameter	Grid Search Values
hidden_layer_sizes	(50,), (100,), (150,)
activation	relu, tanh
solver	adam
alpha	0.0001, 0.001, 0.01
learning_rate	constant, invscaling, adaptive
max_iter	200, 300, 400
early_stopping	True
validation_fraction	0.1

Table 7: HyperParameters Grid for MultiLayer Perceptron

HyperParameter	Grid Search Values
depth	4, 6, 8
learning_rate	0.05, 0.1
iterations	100, 200
l2_leaf_reg	3, 7

Table 8: HyperParameters Grid for CatBoost

HyperParameter	Grid Search Values
num_leaves	31, 63
learning_rate	0.05, 0.1
feature_fraction	0.8, 0.9
bagging_fraction	0.8, 0.9
n_estimators	50, 100
max_depth	5, 7

Table 9: HyperParameters Grid for LightGBM

Appendix C

Fixed Split Setting Grid Search Results

Metric Name	KNN	Random Forest	MLP	CatBoost	LightGBM	EdgeBank
Accuracy	0.826818	0.844531	0.834134	0.840545	**0.848245**	0.682727
Precision	0.905714	0.907253	**0.910542**	0.910754	0.907667	0.625241
Recall	0.731734	**0.789764**	0.638135	0.776245	0.781445	0.701529
F1	0.812684	0.836363	0.822352	0.8293565	**0.840047**	0.696246
ROC AUC	0.810663	0.830457	0.844145	0.828154	**0.834143**	0.636356
Average Precision	0.793650	0.811651	**0.811230**	0.809568	0.810425	0.623422

Figure 10: Grid Search results for Fixed Split Setting and k = 1

Metric Name	KNN	Random Forest	MLP	CatBoost	LightGBM	EdgeBank
Accuracy	0.809491	0.829173	**0.843986**	0.827596	0.834042	0.637430
Precision	0.905714	0.907253	**0.910881**	0.910754	0.907667	0.625241
Recall	0.695180	0.737135	**0.766056**	0.730210	0.747454	0.704277
F1	0.786603	0.813394	**0.832215**	0.810550	0.819806	0.662410
ROC AUC	0.810663	0.830117	0.844785	0.828594	**0.834930**	0.636744
Average Precision	0.783591	0.801534	**0.815945**	0.801306	0.805994	0.589705

Figure 11: Grid Search results for Fixed Split Setting and k = 2

Metric Name	KNN	Random Forest	MLP	CatBoost	LightGBM	EdgeBank
Accuracy	0.827483	**0.850562**	0.784749	0.847490	0.847403	0.648473
Precision	**0.909287**	0.902803	0.907476	0.907836	0.903394	0.638318
Recall	0.731411	**0.789090**	0.638985	0.776928	0.781445	0.701529
F1	0.810707	**0.842125**	0.749924	0.837296	0.838007	0.668432
ROC AUC	0.828471	**0.851194**	0.786248	0.848216	0.848081	0.647928
Average Precision	0.800724	**0.818921**	0.762209	0.817995	0.816343	0.598553

Figure 12: Grid Search results for Fixed Split Setting and k = 3

Metric Name	KNN	Random Forest	MLP	CatBoost	LightGBM	EdgeBank
Accuracy	0.820381	**0.843719**	0.839932	0.838343	0.840909	0.675587
Precision	0.904085	0.897721	**0.907338**	0.905118	0.901556	0.666741
Recall	0.721565	**0.780005**	0.761410	0.760203	0.769621	0.717460
F1	0.802579	**0.834733**	0.827994	0.826355	0.830380	0.691171
ROC AUC	0.821579	**0.844492**	0.840883	0.839290	0.841773	0.675079
Average Precision	0.793240	0.811541	**0.811580**	0.809408	0.810425	0.621322

Figure 13: Grid Search results for Fixed Split Setting and k = 4

Metric Name	KNN	Random Forest	MLP	CatBoost	LightGBM	EdgeBank
Accuracy	0.826818	0.844318	0.834545	0.840000	**0.848182**	0.682727
Precision	0.893514	0.890076	0.894180	**0.895094**	0.893002	0.672255
Recall	0.745266	0.788548	0.761948	0.773219	**0.793959**	0.723174
F1	0.812684	0.836242	0.822785	0.829705	**0.840573**	0.696785
ROC AUC	0.827491	0.844778	0.835144	0.840551	**0.848629**	0.682394
Average Precision	0.794314	0.808459	0.801318	0.806422	**0.812870**	0.625703

Figure 14: Grid Search results for Fixed Split Setting and k = 5

Appendix D

Live Settings Results

Metric Name	KNN	MLP	CatBoost	T3GNN	LightGBM	EdgeBank
Accuracy	0.735003	**0.768822**	0.751416	0.717069	0.618138	0.555416
Precision	**0.768067**	0.765004	0.767233	0.761308	0.724344	0.550788
Recall	0.698968	**0.770592**	0.712186	0.623214	0.566451	0.402921
F1	0.721265	**0.749691**	0.725628	0.675016	0.623861	0.431891
ROC AUC	0.735566	**0.769173**	0.751832	0.717747	0.616001	0.555388
Average Precision	0.706697	**0.733432**	0.719982	0.693158	0.657354	0.544183

Figure 15: Averaged evaluation metrics results for Live Setting and k = 2

Metric Name	KNN	MLP	CatBoost	T3GNN	LightGBM	EdgeBank
Accuracy	0.715889	0.747000	**0.747613**	0.732682	0.610918	0.567166
Precision	0.752669	0.759725	**0.768200**	0.764391	0.728280	0.534266
Recall	0.657453	**0.701657**	0.688794	0.642597	0.533449	0.407116
F1	0.695337	0.717562	**0.718275**	0.688798	0.601970	0.436453
ROC AUC	0.716392	0.747440	**0.747916**	0.733097	0.613743	0.567142
Average Precision	0.693399	0.712351	**0.717873**	0.705635	0.657493	0.551894

Figure 16: Averaged evaluation metrics results for Live Setting and k = 3

Metric Name	KNN	MLP	CatBoost	T3GNN	LightGBM	EdgeBank
Accuracy	0.717511	**0.756197**	0.742013	0.720312	0.622913	0.581944
Precision	0.747435	0.747547	**0.760293**	0.751795	0.717114	0.596679
Recall	0.671587	**0.740099**	0.676775	0.624847	0.605293	0.415850
F1	0.698351	**0.738762**	0.710553	0.676130	0.643367	0.451037
ROC AUC	0.718119	**0.756515**	0.742426	0.720900	0.618220	0.582054
Average Precision	0.689569	**0.724277**	0.713085	0.695018	0.661735	0.563123

Figure 17: Averaged evaluation metrics results for Live Setting and k = 4

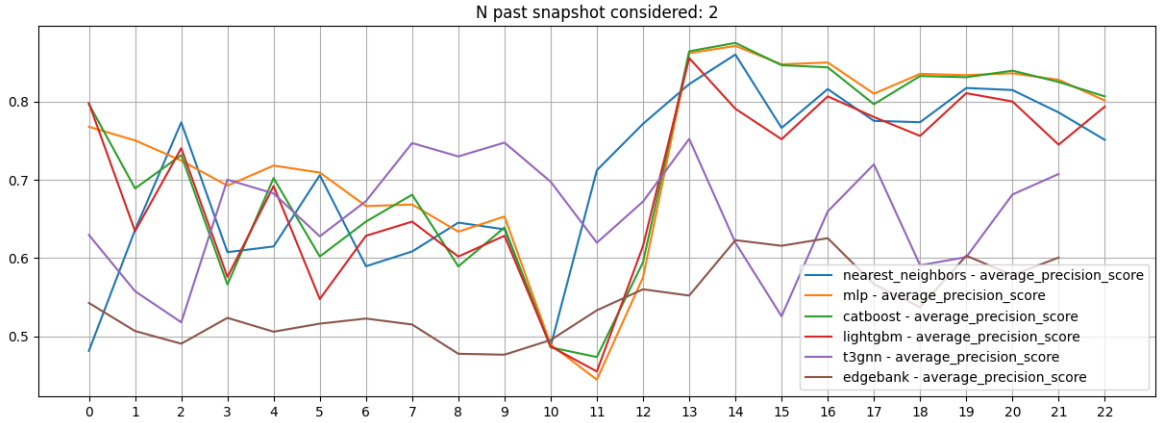

Figure 18: Average Precision Score behavior for 2 past snapshots

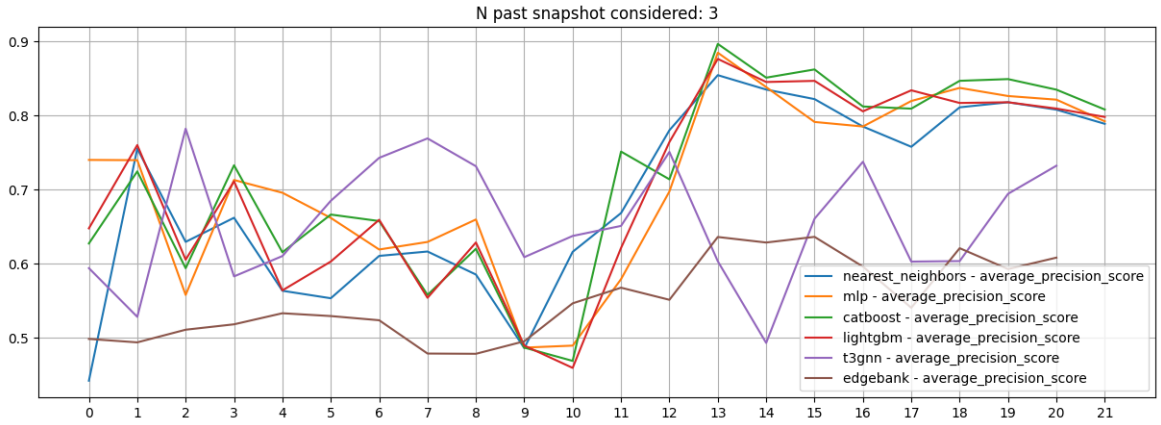

Figure 19: Average Precision Score behavior for 3 past snapshots

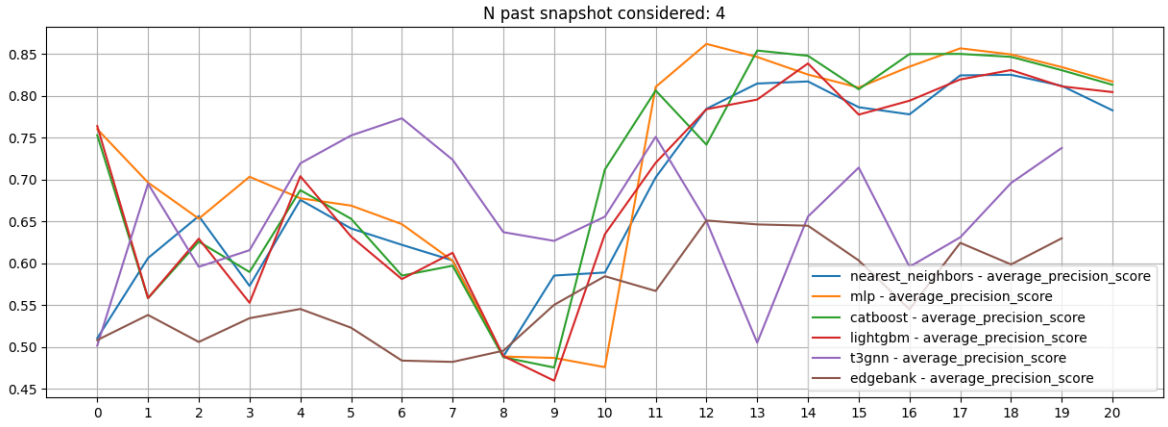

Figure 20: Average Precision Score behavior for 4 past snapshots

Bibliography

[1] Farimah Poursafaei, Shenyang Huang, Kellin Pelrine, and Reihaneh Rabbany. Towards better evaluation for dynamic link prediction, 2022.

[2] Jiaxuan You, Tianyu Du, and Jure Leskovec. Roland: Graph learning framework for dynamic graphs, 2022.

[3] Manuel Dileo, Matteo Zignani, and Sabrina Gaito. Temporal graph networks and sentence embedding for transaction prediction in dynamic web3 social platforms, 2023.

[4] Seyed Mehran Kazemi, Rishab Goel, Kshitij Jain, Ivan Kobyzev, Akshay Sethi, Peter Forsyth, and Pascal Poupart. Representation learning for dynamic graphs: A survey. *Journal of Machine Learning Research*, 21(70):1–73, 2020.

[5] Antonio Longa, Veronica Lachi, Gabriele Santin, Monica Bianchini, Bruno Lepri, Pietro Lio, Franco Scarselli, and Andrea Passerini. Graph neural networks for temporal graphs: State of the art, open challenges, and opportunities, 2023.

[6] Gaito S Ba CT, Zignani M. The role of cryptocurrency in the dynamics of blockchain-based social networks: The case of steemit. 2022.

[7] Aric A. Hagberg, Daniel A. Schult, and Pieter J. Swart. Exploring network structure, dynamics, and function using networkx. In Gaël Varoquaux, Travis Vaught, and Jarrod Millman, editors, *Proceedings of the 7th Python in Science Conference*, pages 11 – 15, Pasadena, CA USA, 2008.

[8] Lada A Adamic and Eytan Adar. Friends and neighbors on the web. *Social Networks*, 25(3):211–230, 2003.

[9] Albert-László Barabási and Réka Albert. Emergence of scaling in random networks. *Science*, 286(5439):509–512, 1999.

[10] Yang Yang, Ryan N. Lichtenwalter, and Nitesh V. Chawla. Evaluating link prediction methods. *Knowledge and Information Systems*, 45(3):751–782, October 2014.

[11] Mu Zhu. Recall, precision and average precision. 09 2004.

This project has been developed in the Connets Lab
https://connets.di.unimi.it/