Università degli Studi di Padova

Dipartimento di Ingegneria dell'Informazione

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

COMPUTER ENGINEERING MASTER'S DEGREE

HIGH PERFORMANCE AND BIG DATA COMPUTING

A GRAPH APPROACH TO SETTLEMENT INSTRUCTIONS STATUS PREDICTION

Supervisor: Francesco Silvestri

External co-supervisor: Enrico Papalini

Graduate student: Luca Crema

ACADEMIC YEAR 2021-2022

Graduation date 05/12/2022

Abstract

In financial post-trade settlement, participants may fail to fulfill their orders to deliver the required cash or securities. These orders are called "settlement instructions" and one of the reasons why they fail could depend on other participants' failures in a chain fashion. Euronext Securities Milan's currently predicts whether an instruction will fail using a neural network model that takes a single instruction as input, lacking to account for external relationships. In this research we modeled settlement instructions as graphs to extract and use the relationships information in the prediction task by applying two different approaches: in the first one we computed node-level metrics on the graph to be used as instructions features, and in the second one we introduced graph neural networks layers to the prediction model. Although the target was to improve the prediction model accuracy over the binary classification task (settles or fails), we used it as a sub-task for the prediction of the daily imbalance, which is the sum of the credits and debits from the successfully settled instructions of a client. We managed to achieve the target for some of the clients we considered, with slight improvements both in classification accuracy and daily imbalance prediction.

Contents

1 Introduction **5**

1.1 Motivation 5

1.2 Subjects covered 6

1.3 Foundations of financial and post-trade concepts 6

1.3.1 Financial and trading concepts 6

1.3.2 Post-trading concepts 8

1.3.3 Settlement instructions 9

1.3.4 Settlement failure 11

1.4 Problem formalization 12

1.4.1 Foundations of binary classification problems 14

1.5 Available data 15

1.6 Review of current approaches 18

1.6.1 The settlement fails prediction approach 18

1.6.2 The liquidity forecast approach 20

1.6.3 The infrastructure supporting the approaches 22

2 Graph modeling **24**

2.1 Participants-centric graphs 25

2.2 Instruction-centric graphs 28

2.3 Other details 30

2.3.1 Amount of data 30

2.3.2 Additional subsets 31

3 Approach 1: features extraction **33**

3.1 Extracted metrics 36

3.1.1 Degrees 37

3.1.2 Closeness & Harmonic centrality 37

3.1.3 Betweenness centrality 40
3.1.4 Pagerank coefficient 42
3.1.5 Clustering coefficient 43
3.1.6 Triad census 44
3.1.7 Communities 45
3.1.8 Node embeddings 48
3.2 Feature selection and source graph 50
3.3 Results 55

4 Approach 2: GNNs **59**
4.1 Introduction to GNNs 59
4.1.1 Message passing framework 59
4.1.2 Basic GNN 60
4.1.3 Graph Convolutional Networks GCNs 61
4.1.4 Graph SAGE 62
4.2 Usage 62
4.3 Models 63
4.4 Evaluation and results 64

5 Conclusions and future works **68**

List of Figures

2.1 Example of graph G and the corresponding line graph $L(G)$ 28

3.1 Conditional Entropy 35
3.2 Traditional framework for ML with graphs 35
3.3 Conditional Entropy 44
3.4 XGBoost feature importances 51
3.5 PP-Score of features 52
3.6 Client 1: Imbalance prediction accuracy of models (higher is better) "GRAPH" is (BIC, Failed) 56
3.7 Client 2: Imbalance prediction accuracy of models (higher is better) "GRAPH" is (BIC, Complete) 57
3.8 Client 3: Imbalance prediction accuracy of models (higher is better) "GRAPH" is (BIC, ISIN) 58

4.1 Example graph 60
4.2 Message passing framework computation 61
4.3 Template structure of our GNN-based models 64
4.4 Comparison of the number of errors between GNN-like models and the naïve over all instructions from April 65
4.5 Client 1: models comparison in the number of errors (lower is better) 65
4.6 Client 2: models comparison in the number of errors (lower is better) 66
4.7 Client 3: models comparison in the number of errors (lower is better) 66

List of Tables

1.1 Binary classification contingency table 14
2 Dataset fields table 15
1.3 Models eval. metrics 19
1.4 Models eval. metrics using instructions countervalues 19

1 Extracted degree metrics 37
2 Extracted Closeness centralities features 39
3 Extracted Betweenness centrality features 41
4 Extracted Pagerank coefficient features 42
5 Extracted Clustering coefficient features 44
6 Triad census features 45
7 Features from community algorithms 48
8 Node2Vec feature vectors 50
9 Graph-features used in next steps 52
3.10 Source graphs 53
11 Clients statistics 55
12 Client 1 56
13 Client 2 56
14 Client 3 57

4.1 GNN-based models evaluation metrics 65
4.2 Client 1: Prod and GNN-based models evaluation metrics 65
4.3 Client 2: Prod and GNN-based models evaluation metrics 66
4.4 Client 3: Prod and GNN-based models evaluation metrics 67

Chapter 1

Introduction

This is a master's thesis based on my work during an internship in *Euronext Securities Milan* (*ESM*) under the supervision of Enrico Papalini from March to May and again from July to November of 2022. During the internship we used graph representations of **settlement instructions**, that *ESM* sends to the European *TARGET2-Securities platform*, to predict their final **settlement status**. In the next introductory pages, we will review some required financial definitions and post-trade concepts, we will formalize the problem we are trying to solve, describe available data, and then overlook the current approach.

1.1 Motivation

> The term **settlement instruction** is a generic term used to describe the (only) mechanism by which trade settlement (the exchange of securities and cash) is initiated between seller and buyer.
> - Simmons, Michael - Securities operations [1]

While most settlement instructions are settled at the first attempt, some take more and others may never settle. Successful settlement of a settlement instruction can depend directly on the behavior of buyer and seller, or it can depend on the success of other instructions through *daisy chains.*

Predicting whether an instruction will settle in the *intended settlement date* is useful information for market participants that are currently looking to further improve their settlement efficiency, and participants' treasuries that require a precise cash estimate to minimize funding and reduce fails. This prediction is already computed by the Data Science team in *ESM* using neural networks. The prediction model takes as input the information about a single settlement instruction and, using a few neuron layers, it outputs the predicted settlement status.

This current approach does not employ the information that *ESM* has about other participants of the settlement environment, which could be used by the prediction models to account for dependencies between settlement instructions. To utilize this knowledge, we modeled settlement instructions as graphs and fed this information to prediction models to improve their accuracy. To accomplish this task we came up with two approaches that both yielded slightly better results for some of the clients we considered, and that we will examine in the next pages.

1.2 Subjects covered

The specific subjects covered by this dissertation are:

- Chapter 1: problem definition, description of the available data, and review of the current approach to settlement status prediction.
- Chapter 2: ways to model settlement instructions as graphs.
- Chapter 3: extraction of *graph metrics* used as prediction model's features.
- Chapter 4: introduction of *graph neural networks* layers to the prediction model.

1.3 Foundations of financial and post-trade concepts

Let us review some elementary *financial* concepts and *post-trade* definitions necessary for understanding the rest of this work.

1.3.1 Financial and trading concepts

A **financial asset**[2] is a liquid asset that gets its value from a contractual right or ownership claim. Cash, stocks, bonds, mutual funds, and bank deposits are all examples of financial assets. Financial assets are more liquid than other tangible

[2]Investopedia. *Financial Asset Definition.* URL: `https://www.investopedia.com/terms/f/financialasset.asp`.

assets, such as commodities or real estate, and may be traded in financial markets. A **security** is a fungible, negotiable financial asset that holds some type of monetary value. A security is universally uniquely identified by a 12 alphanumeric characters string called International Securities Identification Number (**ISIN**). There are two main locations where financial assets are traded: **"regulated markets"** (RM) and **"over-the-counter"** (OTC). *Over-the-counter* trading is done directly between two private parties, and the prices of the trades are usually not made public. In *regulated markets* instead, all trades are public so every trader knows the price of traded assets. A **participant** to a regulated market is an entity that is directly or indirectly identified in such market and is capable to send and receive transfer orders[3]. Examples of *participants* are credit institutions, investment firms, and public authorities. A *participant* is said to be **indirect** when it relies on another participant for market access. A **stock**, also known as *equity*, is a security that represents the ownership of a fraction of the issuing corporation[4]. Units of stock are called "*shares*" which entitles the owner to a proportion of the corporation's assets and profits equal to how much stock they own. One of the primary advantages of owning stocks is the regular payment of **dividend** income. *Ordinary dividends* are a share of a company's profits passed on to the shareholders periodically[5]. **Stock exchanges** are a type of regulated market where shares of **listed** companies are traded. A **stock broker** is a licensed participant of a *stock exchange*, which is authorized to carry on the business of trading securities on behalf of other persons or on their own account. A **corporate action** is an event carried out by a company that materially impacts its stakeholders. Common corporate actions include the payment of *dividends*, stock splits, tender offers, and mergers and acquisitions[6].

[3]European Central Bank (ECB). *Glossary of terms related to Payment, Clearing and Settlement systems.* URL: https://www.ecb.europa.eu/pub/pdf/other/glossaryrelatedtopaymentclearingandsettlementsystemsen.pdf.

[4]Investopedia. *Stock.* URL: https://www.investopedia.com/terms/c/stock.asp.

[5]Investopedia. *Ordinary dividend.* URL: https://www.investopedia.com/terms/c/ordinary-dividends.asp.

[6]Investopedia. *Corporate Action.* URL: https://www.investopedia.com/terms/c/corporateaction.asp.

1.3.2 Post-trading concepts

After a trade in a market is executed the *buyer* must make payment for the securities they purchased, while the *seller* must deliver the security that was acquired. This task is part of the *post-trade* process and the transaction enters the **settlement period**. A trade is defined as **settled** when all the money and the securities are successfully exchanged. A *central securities depository* (**CSD**) is a financial entity that enables securities transactions to be settled and provides custodial services[3]. Each market participant must appoint a domestic or international *CSD* to transfer the securities and accomplish the settlement of trades. *Euronext Securities Milan*[7], formerly *"Monte Titoli"*, is the domestic *CSD* for the Italian financial market. A *central counterparty* (**CCP**) is an entity that interposes itself between the counterparties to the contracts traded, becoming the buyer to every seller and the seller to every buyer, thereby guaranteeing the performance of open contracts. Only one *central counterparty* is authorized in Italy, *Cassa di compensazione e garanzia S.p.A.* (CC&G)[8]. A market is said to be **guaranteed** when the presence of a *central counterparty* is mandatory. *CCPs* are required for *regulated markets*' trades but not in *over-the-counter* trades. Financial institutions are uniquely identified by a *Bank Identifier Code* (**BIC**). A *BIC* comprises a financial institution code (four characters), a country code (two characters), a location code, and, optionally, a branch code. Securities are stored in *securities accounts* (*SAC*) that are owned by a legal entity called **settlement agent** that can be a central bank or a central securities depository, a central counterparty, or other kinds of institution[3].

Settlement agents are uniquely identified by an assigned *BIC* code. Liquidity used for cash payments, in relation to securities settlement, is held in *dedicated cash accounts* (**DCAs**). A **cash settlement agent** is either a central bank or a private bank used to provide cash on behalf of a CSD *participant* to support the settlement of securities; it is the owner of one or more *dedicated cash accounts*. **Trading members** are participants with rights to trade on their own account as well as on

[3]European Central Bank (ECB). *Glossary of terms related to Payment, Clearing and Settlement systems.* URL: https://www.ecb.europa.eu/pub/pdf/other/glossaryrelatedtopaymentclearingandsettlementsystemsen.pdf.

[7]*Euronext Securities Milan.* URL: https://www.euronext.com/en/post-trade/euronext-securities/milan.

[8]Banca d'Italia. *Central counterparty.* URL: https://www.bancaditalia.it/compiti/sispaga-mercati/controparte-centrale/index.html.

account of their clients but have no right to clear and settle such trades themselves. **Netting** is the agreed offsetting of the value of multiple positions or payments by participants, this process involves the calculation of *net settlement positions* and their legal reduction to a (bilateral or multilateral) net amount. For example, if a participant buys 40 shares of a security and sells 20 shares of the same security, the position is a net purchase of 20 shares. Netting is used to reduce settlement, credit, and other financial risks between two or more parties. From a 2022 ESM report on bilateral netting "since July 2021, [the number of] settled instructions for market participants went from 3.4 million to 1.1 million, with a netting ratio of 69%"[9].

1.3.3 Settlement instructions

When a trade enters the *post-trade* process, a pair of **settlement instructions** has to be produced. A *settlement instruction* is an order to absolve a payment obligation or transfer the title to a security. An *instruction* contains all the data required to achieve the settlement of a trade. In general, both parties of a trade will produce their side of a *settlement instruction* and dispatch it to a common settlement system. The two sides are called the "**legs**" and they differ in "securities direction", which can be in delivery (valued "*DELI*") or in reception (valued "*RECE*"), and "cash direction", which can be credit or debit. There exist four main *categories* of instructions based on their securities and cash directions[10]:

- **Delivery versus payment** (**DVP**): when securities are exchanged for money.
- **Delivery with payment** (**DWP**): when securities and money move in the same direction.
- **Free of payment** (**FOP**): when securities are transferred without a payment, eg. corporate actions.
- **Free of delivery** (**FOD**): when money is transferred without the simultaneous exchange for securities, eg. dividend payment or penalties payment.

[9] ESM. *Bilateral netting report - Linkedin.* URL: `https://www.linkedin.com/feed/update/urn:li:activity:6960205218395402241/`.

[10] Clearstream. *Clearstream Banking's TARGET2-Securities Glossary.* 2022. URL: `https://www.clearstream.com/resource/blob/1316800/461f1a121cd02eccac538e760de29042/t2s-glossary-data.pdf`.

Before attempting to process an instruction its two legs must first **match**. *Matching* is accomplished by checking that the two legs' fields have equal (eg. ISIN, quantity, and amount) and opposite (securities and cash directions) values[11]. Not all instructions require both legs (e.g. FOPs), so matching is not always needed.

> After the corresponding instructions have been matched, the system will attempt to settle the transaction. This process involves checking whether the participant must deliver the securities has enough securities in their account and whether the buyer has the funds to pay the seller. If one of the two parties do not have the required securities or funds, the transaction is put on hold and other settlement attempts will be made later (first on the same accounting day and then, if the rules of the system provide for "recycling", over several subsequent days). If the securities or funds are sufficient, the transaction is said to be "settled" and becomes "final".
>
> - [12]

Settlement instructions are usually not settled immediately, instead, their **intended settlement date** is set to be two business days after the date of the trade ($T + 2$ where T is the execution date). A *settlement instruction* is said to be **cross-border** when the two participants appoint different CSDs. To perform *cross-border* settlement, the two CSDs use a fictitious account in their own settlement systems that act as a counterparty to the settlement instruction. These two accounts are named "*Mirror* account" on the investor side and "*Omnibus* account" on the issuer side; they are synchronized using **realignment** instructions.

The heterogeneity of settlement practices between different *CSDs* made it necessary to adopt complex solutions to perform *cross-border* settlement. To overcome this issue *TARGET2-Securities* (**T2S**[13]) was created by the European Central Bank. *T2S* is the common *European securities settlement system*, aiming to reduce settlement risk and increase financial stability by using national or international central bank liabilities for transactions. It has also reduced the costs and complexity of *cross-border* settlement by handling *realignment* instructions generation automati-

[11] *T2S - Matching fields from a message perspective.* 2015. URL: `https://www.ecb.europa.eu/paym/target/t2s/profuse/shared/pdf/insights_on_matching_fields_from_a_message_perspective_t2s.pdf`.

[13] *TARGET2-Securities.* URL: `https://www.ecb.europa.eu/paym/target/t2s`.

cally.

ESM developed a *pre-settlement* platform and *T2S*-interface called **XTRM**. *XTRM* receives regulated market trades and over-the-counter orders from participants and converts them into *settlement instructions* to be sent to *T2S*. It also deals with interposing the *CCP* when needed by splitting trades in two where the *CCP* is one of the two parties (from A-B to A-C, C-B, where A and B are the two parties of the trade and C is the appointed central counterparty). Not all *ESM*'s clients use *XTRM*, as they can access *T2S* directly; these clients are called *directly connected participants* (**DCPs**), while those that use *XTRM* are *indirectly connected participants* (**ICPs**).

1.3.4 Settlement failure

When the settlement of a trade does not succeed, it *fails*.

> A trade is said to **fail** if on the *intended settlement date* either the seller does not deliver the securities in due time or the buyer does not deliver funds. While a relatively low rate of fails can be considered "physiological", high settlement fail rates may result in **"daisy chains"** (a cascading chain of fails), which may degenerate into a *"round robin"* (where the last participant in the chain is itself failing to the first participant), leading to *gridlock* situations.
>
> - "Settlement fails - report on securities settlement systems" - ECB [14]

The ECB article cited above also proceeds to list three possible *reasons* for *settlement failure*:

- **Operational risk**: Human or computer errors, e.g. miscommunication between traders or mistakes in processing by back offices.

- **Liquidity problems**: Unavailability of the asset due to cascading fails, technical conditions in other market segments or *naked short selling*[15].

[15] Investopedia. *Naked Short Selling.* URL: `https://www.investopedia.com/terms/n/nakedshorting.asp`.

- **Lack of incentive to avoid fails**: When the *costs of delivering* the asset are higher than the *cost of failing* to deliver, market participants find it preferable to fail, this phenomena is also called **strategic fail**. This problem has been partially tackled in 2019 with the introduction of the *settlement discipline*[16] framework by the ECB to disincentivize this practice.

If an instruction has *failed* to settle during the current day, it transitions to the next settlement date (a process named **recycling**) to try to settle again. After 60 days of recycling the instruction is removed from the system.

1.4 Problem formalization

It is important to monitor the number of settlement fails to measure the *healthiness* of settlement systems. The document "report on security settlement systems"[14] lists three concrete consequences of settlement fail:

- The parties involved continue to be exposed to **credit risk** or **replacement cost risk**.
- Both parties are exposed to **liquidity risk on the settlement date**: the seller because of lack of the expected cash and the buyer because of the inability to use the expected incoming assets to settle back-to-back transactions due on the same day. If the fail is not promptly resolved, settlement problems may propagate to other transactions, and potentially trigger a disturbance of the smooth settlement process.
- The securities lending markets for the securities subject to the significant volume of fails could also be negatively affected, as **lenders may withhold securities** in the fear that the high fails in that security might *diminish the likelihood of the assets being returned* to them, and this could negatively affect collateral markets. This withholding of scarce securities could in turn contribute to increasing the fail rate and prolonging the fail duration.

[16] *T2S Penalty Mechanism.* 2017. URL: `https://ecsda.eu/wp-content/uploads/2019/01/Annex_I_1_T2S_Penalty_Mechanism.pdf`.

[14] European Central Bank (ECB). "Settlement fails – report on securities settlement systems". In: (2011). URL: `https://www.ecb.europa.eu/pub/pdf/other/settlementfails042011en.pdf`.

To better grasp the extent of the settlement fails problem we consider some recent **official statistics**. The 2020 ESMA report on Trends, Risks and Vulnerabilities (TRV)[17] shows that an average of 2%-4% of the value of all bond trades were unable to settle between 2018-2020, rising to 6%-12% for equities. A more recent 2022 ESMA TRV report[18] exhibits a reduced number of fails in 2021, nevertheless, an average of 2%-3% of the value of bond trades and 5%-8% of the value of equities trades failed to settle.

To prevent the above-mentioned settlement failure consequences the securities settlement system has to be constantly monitored. Predicting whether a trade will fail can help increase settlement performance by providing settlement participants with a preview of trades that are most likely to fail. As a CSD, *Euronext Securities Milan* is in charge of monitoring the settlement performance of its clients, settlement predictions can help earlier detection of voluntary and involuntary wrongdoing. Another employment of the predictions can be found in a service that *Euronext Securities Milan* provides to some of its clients: *cash settlement agents* are provided with an estimate of trades that will settle during the *night-time settlement* (NTS) period; the bank's treasury uses this estimate to precisely allocate money in *dedicated cash accounts* for settlement operations.

The prediction problem can be modeled as a **supervised binary classification** problem and it is currently solved using *machine learning* algorithms. The classification task is performed over the two possible outcomes of settlement *instructions*, which are "**settled**" (securities and money have been exchanged successfully) or "**failed**" (exchange didn't occur). For the binary classification we will consider *settled* instructions to have a **false** value and *failed* instructions to have a **true** value. There exists another possible outcome, that we will consider as *failed*, which is "**partially** settled".

The problem is "supervised" as we have the history of settlement instructions available, and "imbalanced" because we have seen that on average (much) less than 15% of them fail.

[17]European Securities and Markets Authority. "ESMA Report on Trends, Risks and Vulnerabilities - 2020". In: (2020). URL: https://www.esma.europa.eu/sites/default/files/library/esma_50-165-1287_report_on_trends_risks_and_vulnerabilities_no.2_2020.pdf.

[18]European Securities and Markets Authority. "ESMA Report on Trends, Risks and Vulnerabilities - 2022". In: (2022). URL: https://www.esma.europa.eu/sites/default/files/library/esma50-165-2061_trv_1-22_statistical_annex.pdf.

1.4.1 Foundations of binary classification problems

In this section, we will review the notions needed to understand binary classification problems. The objective is to find a "(binary) **classifier**" which is an algorithm that implements a *mathematical function* mapping input data to a truth value. In binary classification there are four possible combinations of *actual* category and *predicted* category:

- **TP** True Positives: predicted true and actually true
- **TN** True Negatives: predicted false and actually false
- **FP** False Positives: predicted true and actually false
- **FN** False Negatives: predicted false and actually true

and they can be plotted in a 2x2 contingency table (1.1).

		Predicted	
		T	**F**
Actual	**T**	TP	FN
	F	FP	TN

Table 1.1: Binary classification contingency table

There are eight ratios that one can compute from this table[19], the two most commonly used are **precision** (positive predictive value) defined as $\frac{TP}{TP+FP}$ and **recall** (true positive rate) defined as $\frac{TP}{TP+FN}$. *Precision* measures how many instructions predicted as failed are actually failed and *recall* measures how many actually failed instructions were predicted as failed. These two metrics are more useful in imbalanced problems over the conventional *accuracy* metric ($\frac{TP+TN}{\text{all instances}}$) as they better capture the quality of classification[20]. A third metric used in binary classification is

[19]Wikipedia. *Binary classification.* URL: `https://en.wikipedia.org/wiki/Binary_classification`.

[20]Rehmsmeier M Saito T. "The Precision-Recall Plot Is More Informative than the ROC Plot When Evaluating Binary Classifiers on Imbalanced Datasets". In: (2015). URL: `https://doi.org/10.1371/journal.pone.0118432`.

the **F1-score**, which is the harmonic mean of *precision* and *recall*

$$F_1 = 2 * \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

it is considered more convenient to work with as one only has to maximize a single score instead of balancing two.

1.5 Available data

Before proceeding it can be useful to review the details of data available to solve the problem. The **dataset** comprises *settlement instructions* in tabular form, they are gathered from *XTRM*, so they mostly cover *ESM*'s clients with indirect access to T2S (*ICPs*). Data considered in this dissertation spans over four years: from the 1th May 2018 to 1th May 2022. In this timespan there are 208,202,110 settlement instructions available, for an average of 206,550 per trading day and for a total size of around 25 GBs. The dataset is updated daily and it is currently stored in an **AWS S3** [21] bucket in *Parquet* format [22]. "**Parquet** is an open source column-oriented data file format designed for efficient data storage and retrieval", one of its advantages over other classic file formats (es. CSV or JSON) is the integrated possibility to selectively load rows with a given value for an index column: in our case, the dataset is indexed by one of the date columns (`dt_business`, see table 1.2).

The following table (table 1.2) presents an explanation of available fields in the dataset.

Table 1.2: Dataset fields table

Property	Description	**Property**	Description
`dt_business`	Date in which the details and state of the instruction have been observed	`id_matching`	Identification code of the pair of instruction legs

Continued on next page

Table 1.2: Dataset fields table (Continued)

`dt_trade`	Date in which the trade was executed	`id_t2s`	Identification code of the single instruction leg
`dt_settl`	Intended settlement date	`id_t2s_ctrp`	
`cd_sec_mov`	Whether securities are to be received or to be delivered by the leg's party	`id_dca`	Identification code for the dedicated cash account of the leg's party
`cd_credit_debit`	Whether cash is to be delivered (debit) or received (credit) by the leg's party	`id_isin`	ISIN of the exchanged security
`qt_quantity`	Quantity of securities to be moved	`am_amt`	Amount of cash to be moved
`id_deli_csd`	BIC code of the deliverer's CSD	`id_rece_csd`	BIC code of the receiver's CSD
`id_deli_pty1_sac_t2s`	Identification code for the T2S security account of the leg's party	`id_rece_pty1_sac_t2s`	Identification code for the T2S security account of the counterparty

Continued on next page

Table 1.2: Dataset fields table (Continued)

`id_deli_pty1_bic`	BIC code of the deliverer's settlement agent	`id_rece_pty1_bic`	BIC code of the receiver's settlement agent
`id_deli_pty2_bic` [1]	BIC code of the deliverer's trading member	`id_rece_pty2_bic` [1]	BIC code of the receiver's trading member
`id_ccp_bic`	BIC code of the central counterparty	`cd_mkt_type`	Acronym of the origin market of the instruction (e.g. `EXCH` for regulated market, `OTCO` for over-the-counter)
`cd_sett`	**Target variable**, whether the instruction has settled		

[1] Filling of the trading member field of instructions is not compulsory for participants, so it is mostly empty.

The dataset also comprises other fields that we will not consider such as the full name of the security or a description of the parties. In general both legs of a settlement instruction are present, although they can be merged as a single instruction because the dataset only contains matched ones.

1.6 Review of current approaches

We will now examine past and current approaches *ESM* employs to perform the prediction.

1.6.1 The settlement fails prediction approach

ESM Data Science team has created a unified model to forecast settlement status for any pending ETFs, Corporate bonds, Government bonds and shares related instructions in T2S. The aim of the model is to forecast the status of a settlement instruction in the next settlement cycle if it has already failed to settle at the intended settlement date. The first version of the model was implemented using XGBoost[23] as classifier because XGBoost dominates structured or tabular datasets on classification and regression predictive modeling problems and it is really fast when compared to other implementations of gradient boosting. The XGBoost library implements the gradient boosting decision tree algorithm: this is an approach where new models are created that predict the residuals or errors of prior models and then added together to make the final prediction. It is called gradient boosting because it uses a gradient descent algorithm to minimize the loss when adding new models. The XGboost model with an accuracy of about 65% was promoted in production. Many experiments were conducted to find a better model, optimizing the pre-processing of the dataset and the hyperparameters selection, using a data set containing only an asset class or all together, comparing different classifier architectures, using Neural networks fully connected, 1-dimensional convolutional neural networks and a combination of models (ensemble techniques with hard and soft voting). Table 1.3 below shows the evaluation metrics calculated by counting the number of instructions belonging to TP, FP, TN, FN classes.
Ensemble Learning with Soft Voting Scheme increases the global accuracy by 5,2% w.r.t. XGBoost in PROD environment. Table 1.4 shows the results of the same model with the evaluation metrics computed as usual, but in this case each instruction is represented by its countervalue (expressed as billion €).
Neural Networks and Ensemble with Soft Voting Scheme were able to predict correctly the 86,2% of the overall amount of the instructions, an increase of 8,5% w.r.t.

[23]Tianqi Chen and Carlos Guestrin. "XGBoost". In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, Aug. 2016. DOI: 10.1145/2939672.2939785. URL: https://doi.org/10.1145%2F2939672.2939785.

Model	Precision	Recall	F1	Accuracy	Δ accuracy %
XGBoost PROD	0,676	0,643	0,636	0,643	-
XGBoost NEW	0,694	0,691	0,69	0,691	4,80%
Neural Net FC	0,689	0,691	0,689	0,689	4,60%
Neural Net CONV-1D	0,694	**0,696**	**0,694**	0,693	5,00%
Hard Voting	0,696	0,694	0,692	0,694	5,10 %
Soft Voting	**0,697**	0,695	**0,694**	**0,695**	**5,20** %

Table 1.3: Models eval. metrics

Model	Precision	Recall	F1	Accuracy	Δ accuracy %
XGBoost PROD	0,815	0,777	0,789	0,777	-
XGBoost NEW	0,854	0,858	0,848	0,858	8,10 %
Neural Net FC	0,859	**0,862**	**0,85**	**0,862**	**8,50%**
Neural Net CONV-1D	**0,86**	**0,862**	**0,85**	**0,862**	**8,50%**
Hard Voting	0,859	0,861	**0,85**	0,861	8,40%
Soft Voting	**0,86**	**0,862**	**0,85**	**0,862**	**8,50** %

Table 1.4: Models eval. metrics using instructions countervalues

XGBoost in PROD environment. *ESM* Data Science team selected the simpler model (Neural network fully connected) to be promoted in production, substituting the XGBoost model.

ESM is not the only company to perform the prediction of the settlement status of instructions, there are others. In June 2021, Deloitte published a report on "Artificial intelligence in post-trade processing"[24] which describe a use case in settlement prediction "Smart Chaser" made by the BNP Paribas (2017): this is an AI trading tool that predicted the likelihood of a delayed "trade matching". BNY Mellon reports in October 2020 to have a machine-learning model to predict settlement fails on US Treasuries bonds, which objective is "to monitor the intraday positions much more closely, to manage down their liquidity buffers and offset the risk of failed settlements". In July 2022 Clearstream launched an AI model[25] to predict the percentage of successful eligibility in settlement by any instruction and share this information with their customers to have them act on instructions that are more likely to fail. Euroclear's "SettlementDrive"[26] is another platform that shows clients the list of the instructions related to them and reports a "settlement fail rating" for securities. To summarise, an improvement in the prediction quality of settlement status would be not only directly useful for ESM's clients, but would also give ESM an advantage over its competitors.

1.6.2 The liquidity forecast approach

One of the services that *ESM* provides to its clients is a forecast of the daily **imbalance**. Treasuries, *dedicated cash accounts* owners, are in charge of allocating a sufficient amount of cash at the beginning of business days to perform successful instructions settlement. This amount of cash is called the *imbalance*, and it is the difference between the sum of credit positions and the sum of debit positions. Not all positions have to be considered in the sum, only those originating from successfully

[24]Deloitte. *Artificial intelligence in post-trade processing.* URL: `https://www2.deloitte.com/content/dam/Deloitte/us/Documents/technology/us-artificial-intelligence-in-post-trade-processing.pdf`.

[25]Clearstream. *Xact Web Portal.* URL: `https://www.clearstream.com/clearstream-en/products-and-services/connectivity-1-/clearstreamxact/xactwebportal/xact-web-portal-settlement-tutorials-3099658`.

[26]SettlementDrive. *EasyFocus.* URL: `https://www.euroclear.com/newsandinsights/en/Format/Webinars/liquidity-drive-and-settlement-drive-am.html`.

settled instructions, as the failed ones won't produce an exchange of money.

$$\text{imbalance} = \sum_{\text{credit settled instr}} instr_{amount} - \sum_{\text{debit settled instr}} instr_{amount}$$

Therefore the machine learning task that *ESM* is trying to solve currently is not a classification task but rather a regression task; still, we will see that the classification is still a relevant topic even in this setting. The first solution was to perform forecast over daily imbalance values, but as stated in *Maria Giuseppina Brunelli*'s Master's thesis "previous research work found that traditional time series forecast, such as ARIMA[27] and ARMA[28] models, were not capable of capturing patterns of the event."[29] So, the *ESM* Data Science team devised a strategy based on the supervised classification of settlement instructions that contribute to the *imbalance* value. To compute the forecast, the instructions' future settlement status is predicted via supervised binary classification, then the value of (predicted) successful ones is summed to produce the final numeric output. The binary classification is currently performed by a neural network using a sequence of *embedding* layers and *dense* layers. In an **embedding** layer output vectors are not computed from the input using any mathematical operation, instead, each integer input is used as the index to access a table that contains a vector representation for every possible input value. Embedding can be only applied to categorical features as the set of values is limited and, usually, known in advance. Categorical input features are fed to an embedding layer and the output is concatenated together with the vector of remaining non-categorical input features. The concatenation is then fed to a variable-sized dense layer followed by a dropout layer and a final fixed-size dense layer with sigmoid activation producing the prediction "probability" output. The final step comprises applying a variable threshold (between 0 and 1) to the probability output to determine the final binary output. This threshold is determined to minimize the imbalance error on the training set by changing the rate of false positives and false negatives, although it reduces the overall classification accuracy of the model. More details about the motivation and the structure of the neural network can be found in *Maria Giuseppina Brunelli*'s Master's thesis[29].

[27]Wikipedia. *Autoregressive integrated moving average model.* URL: `https://en.wikipedia.org/wiki/Autoregressive_integrated_moving_average`.

[28]Wikipedia. *Autoregressive moving average model.* URL: `https://en.wikipedia.org/wiki/Autoregressive%E2%80%93moving-average_model`.

[29]Maria Giuseppina Brunelli. *Cash flow forecasting during night-time settlement cycle as a way to improve settlement efficiency in target2-securities.* 2021.

1.6.3 The infrastructure supporting the approaches

Lastly, we review the structure of the computing system that *ESM* has currently set up to allow data scientists to work on solving this problem. Data is kept in a data lake in the AWS cloud where, as previously mentioned, the storage is provided by S3 (Simple Storage Service)[21], the ingestion is carried out by ATHENA using REDSHIFT as support database , data is transformed in the data lake using Glue Jobs with Spark SQL support triggered by Lambdas, and other data science-related work is performed in SageMaker's Python Jupyter notebooks. We will not analyze all the above-mentioned technologies, but we will overlook some of the SageMaker[30] functionalities and how they are used in *ESM*. SageMaker is a "fully managed service to build, train, and deploy machine learning models for any use case with fully managed infrastructure, tools, and workflows"; some of these tools used by *ESM* are:

- Virtual machines running Python Jupyter that can easily read and write data in S3
- API endpoint to allocate high-performance virtual machines specifically used to perform model training given the values of hyperparameters
- Storage of trained machine learning models that can be used to perform predictions over batches of data via an API

SageMaker provides many more useful functionalities like automatic hyperparameter tuning, although they are currently not available in the AWS Milan region where *ESM* is legally bounded to work.

The training of new ML models is structured in three phases: data preprocessing, model training, and hyperparameters selection. **Data preprocessing** consists in adapting data to the model inputs, which includes the normalization of numerical features and conversion of categorical features to integer values. Original data is loaded from the S3 data lake, processed by the Jupyter notebook SageMaker virtual machines, and then stored back into the data lake to be used by the next phase. **Model training** is performed by the SageMaker training jobs service; each job receives a model, a combination of hyperparameters, and other settings like batch size

[21]AWS. *S3*. URL: `https://aws.amazon.com/s3/`.

[30]AWS. *Sagemaker*. URL: `https://aws.amazon.com/sagemaker/`.

and early stopping patience. As previously mentioned, SageMaker hyperparameters autonomous tuning jobs are currently not available, so a grid-search algorithm is used to explore different hyperparameters combinations. The **selection** of the best model is still subject to research in *ESM*, but as of now the selection is based on a summary score computed on a collection of metrics like the validation set classification accuracy score or the *mean absolute error* (MAE) of the daily imbalance forecast. The review of the current approach concludes the introductory section, where we have seen useful definitions and an overview of the state-of-the-art system.

Chapter 2

Graph modeling

In this section we will provide a detailed explanation of the techniques and tools we used to transform the *tabular settlement instructions datasets into graphs.* Before we can get into the implementation details we briefly review some concepts about graphs. A **graph** is a mathematical model used to represent pairwise relationships between entities. In a graph, entities are represented as **vertices**, also called *nodes*, and relationships are **edges**, also called *arcs*, that link two vertices called *incident vertices.* The more formal definition of graph is

$$G = (V, E) \text{ where } V := \text{set of vertices, } E := \text{set of edges (pairs of vertices)}$$

A graph is said to be **directed** if $(u, v) \in E$ is an ordered pair, otherwise it is **undirected**. If a graph contains multiple edges between the same two nodes (u, v) it is called a **multigraph**. Both vertices and edges can be associated with **feature vectors** (also called *weights* for edges) with details about the entity or the relationship. Graphs can be *disconnected*, meaning that there can be multiple **connected components** $V_1, ..., V_k \subseteq V$ such that there are no edges between any V_i and V_j. One of the possible ways to represent a graph is the **adjacency matrix** A, which is defined as:

$$A_{i,j} = \begin{cases} 1, & \text{if edge } (i,j) \in E \\ 0, & \text{otherwise} \end{cases}$$

The adjacency matrix is symmetric for undirected graphs and asymmetric for directed ones. A path in a graph is an ordered sequence of edges; there can be multiple paths between two nodes (and multiple shortest paths $P_{u,v}$). In a graph the **distance** between two nodes $d(u, v)$ is the length of the shortest path between them;

the maximum distance between any two nodes is called the **diameter** $\Delta(G)$ of the graph.

$$d(u, v) = \min_{p \in P_{u,v}} l(p) \tag{2.1}$$

$$\Delta(G) = \max_{u,v \in V} d(u, v) \tag{2.2}$$

In general, the graph approach is used when the data is well suited to be represented as relationships and when we believe that those relationships contain valuable information to solve the problem. In our case settlement instructions can be modeled as relationships between market participants (first modeling approach) or entities related *by* market participants (second modeling approach), and, for the reasons in Chapter 1.3.4, we *strongly believe* that there exist dependencies between them that can help the prediction of the settlement status.

Before proceeding to the two modeling approaches we will look at the tools we used. The main ETL (Extract, Transform, and Load) tool is the widely-used Pandas library[31] that we used to load the tabular data in Parquet format from S3.

```python
import pandas as pd
import boto3

region_name = "eu-south-1"
bucket_name = ... # S3 bucket name
remote_filename = "data/dt_business=2020-02-01/"
# Define a client to connect to S3
s3_client = boto3.client("s3", region_name=region_name)
# Using the S3 client, perform a request to read a file given its filename
obj = s3_client.get_object(Bucket=bucket_name, Key=remote_filename)
# Finally, read the contents of the response as parquet format
dataframe = pd.read_parquet(io.BytesIO(obj["Body"].read()))
```

Listing 2.1: Load data from S3 in pandas dataframe

2.1 Participants-centric graphs

The first and most straightforward approach to graph modeling is based on the following idea: use the market participants as set of nodes and the settlement in-

[31] *Pandas Python library.* URL: https://pandas.pydata.org/.

structions as edges.

$$\begin{gathered} G = (V, E) \\ \text{where } V = \{\text{market participants}\} \\ \text{and } E = \{(u, v) : u \text{ and } v \text{ are two participants of a settlement instruction}\} \end{gathered}$$

In this approach edges can also be directed and weighted using the orientation of movement of securities or money in each instruction; for this work we arbitrarily decided to direct edges with the same orientation as securities movement: *from deliverer to receiver*. The resulting graph is then a multi-di-graph, as there can be multiple edges between two participants and, as previously said, edges are directed.

One of the standard data structures to represent a graph is the **edge list**. It is a two-column (or more if edge weights are included) matrix where rows represent edges as pairs of start vertex and end vertex stored respectively in the first column and second column. We can employ the inverse of this approach to produce graphs starting from the available tabular data. We need to select two columns to identify participants and consider the resulting two-columned table as an edge list. Two additional columns, amount and quantity, can be also selected as edge weights.

There are multiple choices of columns that identify different "functionalities" of the two parties of an instruction, we will see the details for two of them:

- **SAC**: identifies a single securities account (in T2S). Provides higher specificity for the sources and destinations of the securities account, gives more information when a settlement agent owns multiple accounts, and works under the assumption that they are independent of one another.

- **BIC**: identifies the settlement agent, owner of the securities account where securities are stored. Lowers the power to distinguish single securities account activities, but better performs under the assumption that securities accounts of the same settlement agent are dependent.

- **BC**: A variation of the BIC is the Bank Code (*BC*) that comprises only the first 4 digits of the BIC code, removing the additional specificity about the branch of the settlement agent; assumes that branches of the same settlement agent share responsibilities during settlement.

These three choices differ in specificity of identification and each works best under the respective listed assumptions.

A positive property of this approach is the limited number of nodes, independent from the choice of identification field: although the numbers vary depending on the considered time-frame, on average there are less than 1000 securities accounts, less than 300 different settlement agents and fewer than 200 bank entities (considering all branches as one entity). This means that as the number of instructions increases, the number of nodes remains virtually constant.

We will now proceed to present some of the code we used to implement this approach. The following code shows how, given the source dataset as pandas dataframe, we employ the networkx's function `from_pandas_edgelist()` to create our graph. In this example we build different graphs using the three different participant-identifying fields, then store the produced graphs in a dictionary.

```python
import networkx as nx

identifying_fields: dict[str, tuple[str, str]] = {
    "sac": ("id_deli_pty1_sac_t2s", "id_rece_pty1_sac_t2s"),
    "bic": ("id_deli_pty1_bic", "id_rece_pty1_bic"),
    "bc": ("id_deli_pty1_bc", "id_rece_pty1_bc"),
}

graphs: dict[str, nx.MultiDiGraph] = {}
for identifier, fields in identifying_fields.items():
    graph : nx.MultiDiGraph = nx.from_pandas_edgelist(
        dataframe,
        source=fields[0],
        target=fields[1],
        edge_attr=['am_amount', 'qt_quantity'],
        create_using=nx.MultiDiGraph
    )
    graphs[identifier] = graph
```

Listing 2.2: Networkx graphs from pandas edgelist using different participant identifying fields.

To wind up, we have seen that the advantages of this approach are the previously mentioned easiness in transforming the available tabular data as a graph by considering the data as an edge list, and the restricted number of nodes independently of the chosen node-identifying field.

2.2 Instruction-centric graphs

In some applications it is more convenient or is required to have the classification target, in our case the settlement instructions, as nodes of the graph. A different and less-immediate approach to the graph modeling consists of considering the *settlement instructions as nodes* that are *linked if they share one of the two participants.*

$$
\begin{gathered}
G = (V, E) \\
\text{where } V = \{\text{settlement instructions}\} \\
\text{and } E = \{(u, v) \text{ unordered} : u \text{ and } v \text{ are nodes that share a participant}\}
\end{gathered}
$$

Note that in this case there could still be multiple edges between two nodes if they share both participants and that directing edges would not have a meaning, so they are undirected. The resulting graph in this case is not a multi-di-graph but just a multi-graph. One possible way to obtain this graph is by computing the *line graph* of our previously-shown participant-centric graph. "The **line graph** $L(G)$ of a graph G is constructed in the following way: for each edge in G, make a vertex in $L(G)$; for every two edges in G that have a vertex in common, make an edge between the corresponding vertices in $L(G)$"[32]. Figure 2.1 shows an example of line graph computation.

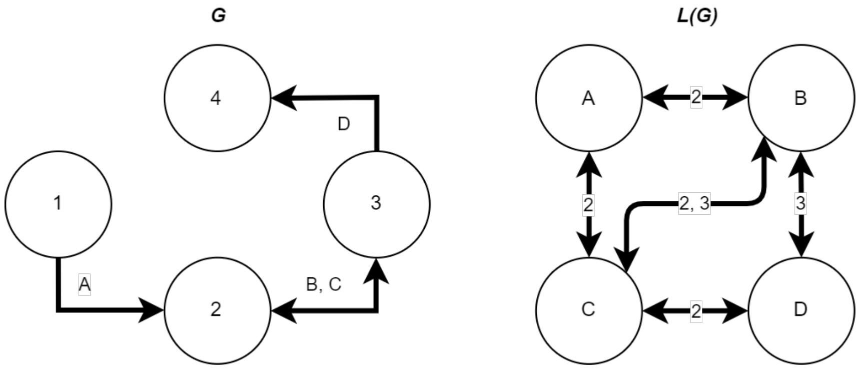


Figure 2.1: Example of graph G and the corresponding line graph *L(G)*

To understand how large the line graph would be we refer to one of its properties: "for a graph G with n vertices and m edges, the number of vertices of the line graph

[32]Wikipedia. *Line graph.* URL: https://en.wikipedia.org/wiki/Line_graph.

$L(G)$ is m, and the number of edges of $L(G)$ is half the sum of the squares of the degrees of the vertices in G, minus m". So in our case, given the original participant-centric graph $G = (V, E)$ and its line graph $G_{LG} = (V_{LG}, E_{LG})$, we have:

$$|V_{LG}| = |E|, |E_{LG}| = \sum_{v \in V} deg^2(v) - |E|$$

From this property we deduce that this instruction-centric graph is much larger in size than the previous one, so it will require some planning about its storage.

Let us again go over some of the code, and see how we tackled this approach. As discussed, there are two steps to reach the target graph: first we transform the data from a tabular dataset into a participant-centric graph, and secondly we apply a function that maps the graph to its line graph. To implement the line graph transformation function we started from the networkx library's implementation; in version `2.8.4` the implementation does not propagate the edge attributes from of G to the vertices of $L(G)$, so we added that functionality to our implementation in `custom_library.line_graph(graph: nx.Graph)`.

```python
import networkx as nx
import custom_library as extra

participants_graph : nx.MultiDiGraph = nx.from_pandas_edgelist(
    dataframe,
    source="id_deli_pty1_sac_t2s",
    target="id_rece_pty1_sac_t2s",
    edge_attr=['am_amount', 'qt_quantity'],
    create_using=nx.MultiDiGraph
)

instructions_graph = extra.line_graph(participants_graph)
```

Listing 2.3: Line graph using a participants-centric graph where participants are identified by T2S security account code

To summarize: although this modeling approach is less intuitive and requires extra attention to memory space occupation, we will see that having the prediction target as nodes helps the GNN model to better compute predictions (Section 4).

2.3 Other details

In the previous sections we have seen two possible graph modeling approaches, but there exist additional considerations that we need to account for.

2.3.1 Amount of data

The first problem we address is the amount of data that we should consider to produce a graph, with "settlement instruction" as our unit measure. The two main points that must be addressed are the **partitioning choices**, informally which settlement instructions are deemed logical to be chosen together to build a graph, and the **feasibility** in terms of *time complexity* and *space occupation*.

Although considering all instructions to a graph is the most informative choice, sometimes it is easier to extract information by partitioning the data and introducing a bias. These *partitioning choices* are usually based on settlement instructions having a common value for one (or more) features; an indication of what could be the best feature comes from the frequency of update of data. It was previously mentioned that the state of settlement instructions is *snapshotted* once a day during the data ingestion; to naturally avoid duplicates it makes sense to group instructions by `dt_business`, so we partitioned the dataset by date. It is worth mentioning that longer periods were considered (week, month, and year), removing duplicates by considering only the last available snapshot per instruction, but the additional information did not balance the information from removed intermediate states and the additional difficulty in handling the extra data.

In terms of *feasibility*, the two modeling approaches differ greatly. The *participant-centric* approach clearly has a linear space complexity relative to the tabular dataset it originates from; in practice partitions of settlement instructions by day, week and month are able to fit the computing instances memories, while larger timeframes do not always fit (crashing the instances). Under the assumption that the number of vertices (participants) is limited to a maximum the time complexity of the networkx's library function `nx.from_pandas_edgelist` it is linear in the number of edges.

On the other hand, the *instruction-centric* graph has a much-worse space complexity as the number of edges is quadratic in the number of instructions; we can compute an approximation by plugging in the average degree $\bar{deg}$ in the $|E_{LG}|$ equation.

$$\bar{deg} = \frac{1}{|V|}\sum_{v \in V} deg(v) = \frac{|E|}{|V|} = \frac{200.000}{200} = 1000$$

$$|E_{LG}| \approx (|V| \cdot \bar{deg}(v)^2) - |E| = (200 \cdot 1000^2) - 200.000 \approx 200.000.000$$

The large number of edges is not only a problem of space, but most importantly of time taken to compute the line graph. In practice we were not able to work with the complete set of instructions from a day (whether there were 200.000 of them or 60.000) as it either took too long to compute or the memory became full and the instance crashed. To avoid these problems we partitioned the daily dataset into groups of at most 10.000 instructions; each partition takes a couple of minutes to be converted to a graph, so a whole day takes less than 20 minutes. By partitioning the originally ordered dataset of instructions we might be introducing a bias based on the proximity of instructions in the order, so we apply a permutation before separating the data. The resulting instruction-centric graphs have 10.000 vertices and around 10.000.000 edges.

In this chapter we have seen the rationale of the choices for the main partitioning techniques we applied, in the next one we will see some additional logic partitioning that was used to extract more specific data.

2.3.2 Additional subsets

In the previous chapter we discussed one of the possible partitioning choices that were applied to most of our work; in this chapter we will see additional partitioning choices that we tried to extract different meaningful data. The partitionings we will see are based on the following financially-based assumptions:

1. When there is a dependency between instructions it is mostly between instructions processing the **same security**.
2. Instructions that did not settle can depend on other **failed instructions** in a domino effect.

The first assumption suggests applying an additional partition of the daily settlement instruction dataset by security's isin. The second suggestion implies that considering failed-only instructions could provide additional information for our purposes. These two extra partitioning choices will be used on our first approach (chapter 3: features extraction).

This section concludes chapter 2, in the next two chapters we will see how we used the different graphs we described to compute the final prediction.

Chapter 3

Approach 1: features extraction

In this chapter we will see one of the two approaches we used to exploit the information that graph-modeling of settlement instructions can provide to improve on the classification task. One of the possible reasons why the current classification model employed by ESM has a limited performance could be the lack of meaningful features in the available dataset – most categorical features have repeating values over the instructions, so they usually convey little information; we will confirm this theory later when we will see how much importance models give to input fields. One of the possible measures of the meaningfulness of some data is entropy and conditional entropy to the target field.

> In information theory, the **entropy** of a random variable is the average level of "information", "surprise", or "uncertainty" inherent to the variable's possible outcomes".
> - Entropy - Wikipedia

Entropy is computed as $H(X) = -\sum_{x \in X} p(x) log\, p(x)$ and intuitively it measures how many bits are required to represent the random variable outcomes.

> Conditional entropy $H(Y|X)$ quantifies the amount of information needed to describe the outcome of a random variable Y given that the value of another random variable X is known.
> - Conditional Entropy - Wikipedia

To compute the conditional entropy $H(Y|X)$ we resort to the chain rule property that states $H(Y|X) = H(X,Y) - H(X)$ where $H(X,Y)$ is the joint entropy; we translated this formula in the following lines of Python code (Listings 3.1 and 3.2).

```python
import numpy as np

def entropy(Y):
    _, count = np.unique(Y, return_counts=True)
    prob = count/len(Y)
    return -np.sum(prob*np.log2(prob))
```

Listing 3.1: Python code to compute entropy

```python
def jEntropy(Y,X):
    YX = np.c_[Y,X].astype(str)
    return entropy(YX)

def cEntropy(Y, X):
    return jEntropy(Y, X) - entropy(X)

for column in df.columns:
    value = cEntropy(df[column], df.cd_sett)
```

Listing 3.2: Python code to compute joint and conditional entropy

By using a data sample from the business week of 21/03 - 25/03 we computed the **empirical** conditional entropy of the target column `cd_sett` to each of the columns in the dataset, and we obtained the results in Figure 3.1. These results can be interpreted as how informative the variable is in determining the settlement status: amount and quantities have many different values across instructions, so by knowing these two values one can easily find the relative (or more than one) instruction in the history and determine its status; although very informative, amount and quantity are prone to overfitting as newer instructions rarely have the same amount and quantity (in case of large quantities). The most informative not-as-prone to overfitting column in the figure is `id_isin`, identifier for the security exchanged, indicating that in the dataset we considered there are securities that are more likely to fail than others. Another thing one can notice is the progression of specificity of participants' identifiers: the security account (`sac_t2s`) is the most specific, the BIC code (`bic`) is a little less specific, and the least specific is the bank code (`bc`); this supports our specificity theories in Chapter 2. One last detail is that the cash movement (`cd_credit_debit`) appears more informative than the securities movement

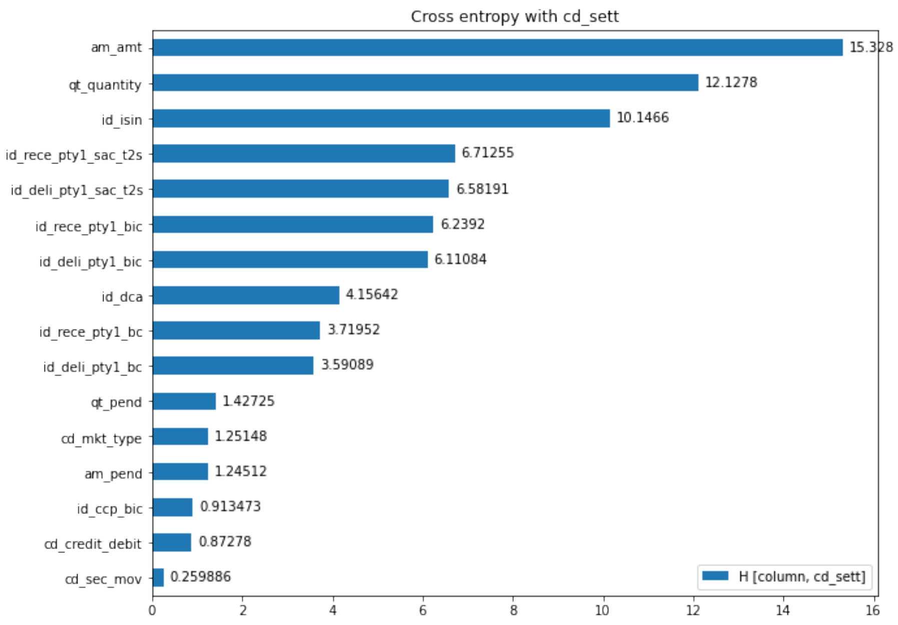


Figure 3.1: Conditional Entropy

(`cd_sec_mov`).

As we have seen, the successful settlement of instructions can depend on chains of relationships between the participants – this information is available to *ESM* but it is currently not exploited by the prediction models. We tried to solve the problem of lack of features and incorporate relationship information by applying the **traditional framework** for machine learning with graphs (Figure 3.2).

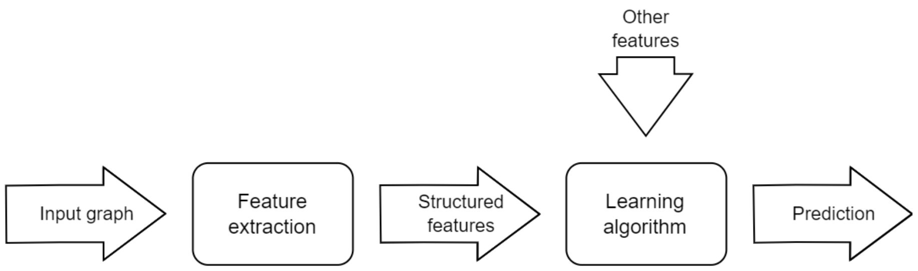


Figure 3.2: Traditional framework for ML with graphs

The feature extraction step consists in computing node-level metrics and graph-level metrics on the input graph, to produce feature vectors that can be concatenated to the input features of the learning algorithm. In this first approach we will mainly work on the feature extraction phase of the framework, by carefully choosing meaningful and feasible features to be extracted from some of the graphs we have previously modeled in chapter 2. The structured features we extract will be joined with the settlement instructions features and together they will be fed to a machine learning model having a similar structure as the one that is currently used to perform the prediction by *ESM*. In our experiments we have only worked with the first kind of graph-modeling we have seen, the participant-centric one, as it was our first choice and the easier to manage between the two choices; in future research we plan to try to consider the instruction-centric kind too.

3.1 Extracted metrics

In this section we will overlook the definitions of the metrics we extracted and the algorithms we used to compute them. The following is a list of the categories of metrics we considered:

- **Weighted degree**
- **Weighted closeness centrality**
- **Weighted betweenness centrality**
- **Harmonic centrality**
- **Pagerank coefficient**
- **Triad counts**
- **Clustering coefficient**
- **Girvan-Newman communities**
- **Louvain's communities**
- **Node embeddings**

All of the previously listed metrics are node-level, during my internship we did not consider graph-level ones. Let us continue and have a detailed view of each and one of them.

3.1.1 Degrees

In graph theory the **degree** of a vertex is the number of edges incident to it. If the considered graph is directed we also have the notion of **in-degree**, which is the number of edges directed to the vertex, and **out-degree**, which is the number of outgoing edges. The **weighted degree** of a vertex in a weighted graph is the sum of the weights in the incident edges – note that if the graph is also directed we have **weighted-in-degree** and **weighted-out-degree**.

$$deg_{\mathrm{w}}(v) = \sum_{e \,\in\, \delta(v)} w_e$$

The degree of every vertex can be easily computed in $O(|E|)$ time and $O(|V|)$ space by iterating over the input edge list and updating a dictionary of type `{vertex: counter}`. It would be an algorithm well-suited for distributed and streaming computation (similar to word count), although in our case the size of the input makes it more convenient to be computed directly. Directed and weighted degrees can be computed similarly by accounting for direction and edge weights.

For our purposes we computed the nine degree-type metrics you can see in Table 3.1.

Table 3.1: Extracted degree metrics

Feature	Description	**Feature**	Description
`degree`	Vertex degree	`sum_amount`	$\sum_{e \,\in\, \delta(v)} \text{amount}_e$
`in_degree`	In-degree	`in_amount`	$\sum_{e \,\in\, \delta^{+}(v)} \text{amount}_e$
`out_degree`	Out-degree	`out_amount`	$\sum_{e \,\in\, \delta^{-}(v)} \text{amount}_e$
`sum_quantity`	$\sum_{e \,\in\, \delta(v)} \text{quantity}_e$		
`in_quantity`	$\sum_{e \,\in\, \delta^{+}(v)} \text{quantity}_e$		
`out_quantity`	$\sum_{e \,\in\, \delta^{-}(v)} \text{quantity}_e$		

3.1.2 Closeness & Harmonic centrality

> In a connected graph, the **closeness centrality** of a vertex is a measure of centrality, calculated as the reciprocal of the sum of the distance (shortest path) between the vertex and every other vertex in the graph.
> - Closeness centrality - Wikipedia

We will refer to closeness centrality as the Wasserman-Faust[33] formulation of the centrality, which accounts for multiple connected components and applies normalization.

$$C(v) = \frac{n-1}{N-1} \cdot \frac{n-1}{\sum_{u \in C_G(v)} d(v,u)}$$

Where G is the graph, $N = |V|$, $C_G(v)$ is the connected component of vertex v, $n = |C_G(v)|$ number of vertices reachable by v and $d(v,u)$ is the distance (shortest path) from vertex v to vertex u. In a weighted graph the distance function $d(v,u)$ will account for the weight of the edges when computing the shortest path. Although the centrality measures are intuitively more informative when computed on the MultiDiGraph, we also computed them on the undirected version to have more features to test. To compute the closeness centrality we did not use an approximation algorithm (Eppstein-Wang[34]) as the exact computation algorithm was deemed fast enough for our purposes. We used the networkx implementation of the algorithm[35] which is similar to the following code (Listing 3.3).

[33]S. Wasserman and K. Faust. *Social Network Analysis: Methods and Applications.* pg. 201. 1994.

[34]David Eppstein and Joseph Wang. "Fast Approximation of Centrality". In: (2000). DOI: `10.48550/ARXIV.CS/0009005`. URL: `https://arxiv.org/abs/cs/0009005`.

[35]Networkx. *Closeness centrality.* URL: `https://networkx.org/documentation/stable/reference/algorithms/generated/networkx.algorithms.centrality.closeness_centrality.html`.

```python
def closeness_centrality(G, weight: str = None):
    closeness_dict : dict[node, float] = {}
    for v in G.nodes:
        # Compute the shortest paths to all reachable nodes from v
        distances : dict[node, float] = shrt_paths(G, v)
        sum_dist : float = sum(distances.values())
        cc : float = 0
        if sum_dist > 0.0:
            cc = (len(distances)-1)/(len(G.nodes)-1)
            cc = cc * (len(distances)-1)/sum_dist
        closeness_dict[v] = cc
    return closeness_dict
```

Listing 3.3: Python code to compute exact closeness centrality

Another approach to multiple connected components is the **Harmonic centrality**, which is similar to closeness centrality but requires the distance function $d(v, u)$ to be equal to inf when there is no path between u and v, thus $\frac{1}{d(v,u)} = 0$.

$$C_H(v) = \sum_{u \neq v} \frac{1}{d(v, u)}$$

Twelve features were extracted using closeness and harmonic centralities, six on the MultiDiGraph and six on the undirected version of the graph (Table 3.2).

Table 3.2: Extracted Closeness centralities features

Feature	Description	**Feature**	Description
Closeness centrality			
`c_centrality`	Unweighted directed	`c_centrality_u`	Unweighted undirected
`amount_c_centrality`	Amount-weighted directed	`amount_c_centrality_u`	Amount-weighted undirected
`quantity_c_centrality`	Quantity-weighted directed	`quantity_c_centrality_u`	Quantity-weighted undirected

Continued on next page

Table 3.2: Extracted Closeness centralities features (Continued)

Harmonic centrality			
`h_centrality`	Unweighted directed	`h_centrality_u`	Unweighted undirected
`amount_h_centrality`	Amount-weighted directed	`amount_h_centrality_u`	Amount-weighted undirected
`quantity_h_centrality`	Quantity-weighted directed	`quantity_h_centrality_u`	Quantity-weighted undirected

3.1.3 Betweenness centrality

The betweenness centrality, similarly to closeness centrality, is a vertex centrality measure based on shortest paths. The betweenness centrality $C_B(v)$ of a vertex $v \in G$ is defined as the fraction of shortest paths between any pair of nodes in the graph that pass through node v.

$$C_B(v) = \sum_{s,t \in V} \frac{\sigma(s,t|v)}{\sigma(s,t)}$$

Where $\sigma(s,t)$ is the number of shortest paths between vertices s and t, and $\sigma(s,t|v)$ is the number of shortest paths between s and t that include vertex v. Once again we used the networkx implementation of the algorithm[36] for exact computation (similar to Listing 3.4).

[36]Networkx. *Betweenness centrality.* URL: `https://networkx.org/documentation/stable/reference/algorithms/generated/networkx.algorithms.centrality.betweenness_centrality.html`.

```python
def betweenness_centrality(G, weight: str = None):
    # Dictionary that counts how many shortest paths pass through a
    node
    passing_paths : dict[node, int] = defaultdict(0)
    # Counter of how many shortest paths there are in G
    n_paths = 0
    for s in G.nodes for t in G.nodes:
        # List of shortest paths from s to t
        paths : list[list[node]] = shortest_paths(G, s, t)
        n_paths += len(paths)

        for path in paths:
            for node in path:
                passing_paths[node] += 1

    betweenness_dict : dict[node, float] = {}
    for node in G.nodes:
        betweenness_dict[node] = passing_paths[node]/n_paths

    return betweenness_dict
```

Listing 3.4: Python code to compute exact betweenness centrality

The networkx library provides an approximate computation algorithm for betweenness centrality but we did not make use of it as the time for the exact computation (less than a minute per graph) was feasible for our purposes. The six features in Table 3.3 were extracted using this metric, three on the MultiDiGraph and three on the undirected version of the graph (Table 3.3).

Table 3.3: Extracted Betweenness centrality features

Feature	Description	**Feature**	Description
`b_centrality`	Unweighted directed	`b_centrality_u`	Unweighted undirected

Continued on next page

Table 3.3: Extracted Betweenness centrality features (Continued)

`amount_b_centrality`	Amount-weighted directed	`amount_b_centrality_u`	Amount-weighted undirected
`quantity_b_centrality`	Quantity-weighted directed	`quantity_b_centrality_u`	Quantity-weighted undirected

3.1.4 Pagerank coefficient

The Pagerank coefficient[37], known for its application in Google Search, measures the importance of a vertex based on incoming edges. It is based on the following simplified equation:

$$PR(v) = \sum_{u \in \delta^-(v)} \frac{PR(u)}{|\delta^+(u)|}$$

where $PR(v)$ is the Pagerank coefficient. As one can see the definition could be recursive depending on the shape of the graph (eg. if u has both an inbound edge and outbound edge with v, $PR(u)$ would require $PR(v)$, which we see in the equation that requires $PR(u)$). For this reason computation is carried out in multiple iterations until the coefficient values converge or the number of iterations reaches a threshold. As implementation we used the networkx's library one[38], and given its complexity we will not transcribe the code.

Table 3.4 reports the features extracted using this metric.

Table 3.4: Extracted Pagerank coefficient features

Feature	Description
`pgrank`	Unweighted directed
`pgrank_amount`	Amount-weighted directed
`pgrank_quantity`	Quantity-weighted directed

[37]Lawrence Page et al. *The PageRank Citation Ranking: Bringing Order to the Web.* Technical Report 1999-66. Previous number = SIDL-WP-1999-0120. Stanford InfoLab, Nov. 1999. URL: `http://ilpubs.stanford.edu:8090/422/`.

[38]Networkx. *Pagerank coefficient.* URL: `https://networkx.org/documentation/stable/_modules/networkx/algorithms/link_analysis/pagerank_alg.html`.

3.1.5 Clustering coefficient

Another metric we considered is the **local clustering coefficient**, that measures how much connected nodes are between them. The metric is based on the ratio between actual number of "triplets" that a vertex forms with any two pair of neighbors and the potential number of them. The number of potential triangles a vertex v can form with its neighbours is $deg(v) \cdot (deg(v) - 1)$, while the actual number is the cardinality of the set of neighbors that have an edge between them, $|\{e_{u,w} : u, w \in \mathcal{N}_v, e_{u,w} \in E\}|$.

$$LCC(v) = \frac{|\{e_{u,w} : u, w \in \mathcal{N}_v, e_{u,w} \in E\}|}{deg(v) \cdot (deg(v) - 1)}$$

We only computed the clustering coefficient for the undirected version of the graph because we could not find a clear definition and utility of it on multi-graphs. The algorithm implementation is pretty straightforward:

```python
def lcc(G):
    # Counter of how many triangles a vertex actually forms
    triangles : dict[node, int] = defaultdict(0)
    for v in G.nodes:
        for u in G.neighbors(u):
            for w in G.neighbors(u):
                if (u, w) in G.edges:
                    triangles[u] += 1

    # Compute the local clustering coefficients
    lccs : dict[node, float] = {
        node: actual_triangles/(G.degree(u) * (G.degree(u) - 1))
        for node, actual_triangles in triangles.items()
    }

    return lccs
```

Listing 3.5: Python code to compute local clustering coefficient

Table 3.5 contains the three features extracted.

Table 3.5: Extracted Clustering coefficient features

Feature	Description
`cc_coeff`	Unweighted undirected
`amount_cc_coeff`	Amount-weighted undirected
`quantity_cc_coeff`	Quantity-weighted undirected

3.1.6 Triad census

A **triad** is defined (by Wasserman, Faust[39]) as one of the possible 16 settings of three nodes that can appear in a directed graph, Figure 3.3 contains a visualization and the relative assigned codes. The **triad census** is a graph-level metric that

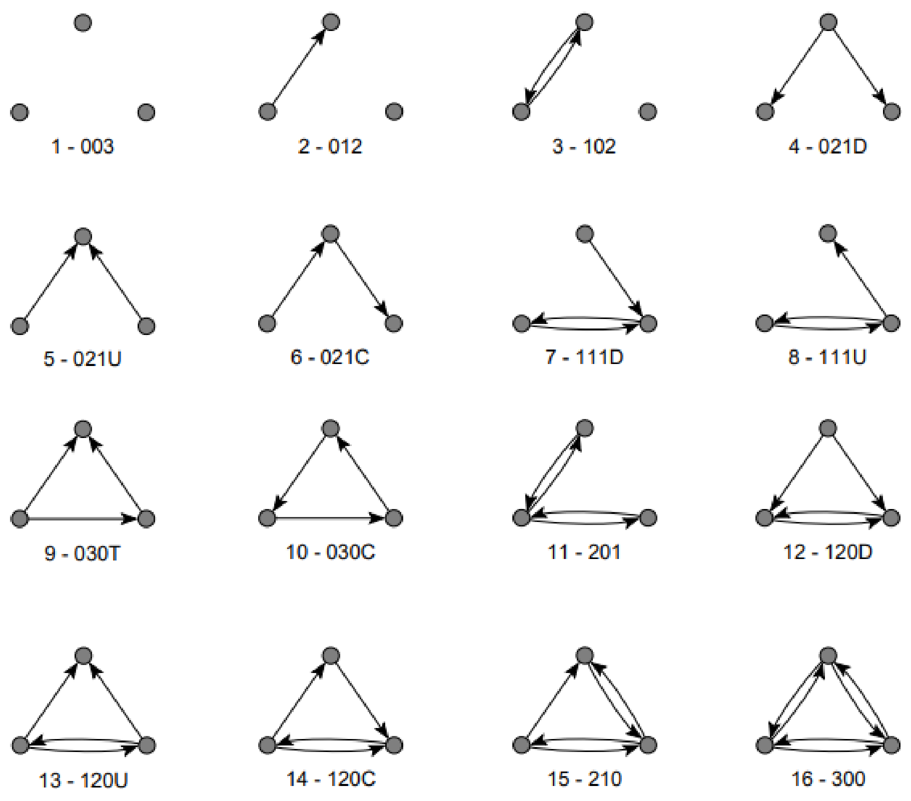


Figure 3.3: Conditional Entropy

consists in the count of how many of the 16 possible types of triads are present in a directed graph. We included the triad census in our features because we believe that

[39]Faust Wasserman. "Social Network Analysis: Methods and Applications". In: (1994). DOI: `https://doi.org/10.1017/CBO9780511815478`.

it can provide us some of the information about dependencies between instructions and participants that the settlement environment has. To compute the triad census we refer to Vladimir Batagelj and Andrej Mrvar's subquadratic algorithm[40] which is implemented in networkx's under `networkx.algorithms.triads.triadic_census`. The algorithm they proposed is able to compute the census for a given subset of V, and we used this capability to transform the triad census in a node-level metric by computing it for each node in the graph. Table 3.6 reports the features extracted with this method.

Table 3.6: Triad census features

Feature	Description
`triad_003`	Count of triads of type 003 the node is part of.
$\vdots$	$\vdots$
`triad_300`	Count of triads of type 300 the node is part of.

3.1.7 Communities

The task to find **graph communities** consists in partitioning V into (non-overlapping) groups of vertices that are "similar" between them and "different" from vertices in other groups. There can be many definitions of the similarity and dissimilarity functions, but the main intuition is that similarity between vertices is represented by the edges. We used two of the availabe community-finding algorithms: the Girvan-Newman algorithm and the Louvain's algorithm.

The idea behind the **Girvan-Newman**[41] algorithm is to iteratively remove the most "central" edge in the graph until we reach an acceptable number of partitions. Various definitions of centrality for edges can be used, but the most common one is **link betweenness**: similarly to betweenness centrality, let $\sigma_{s,t}$ be the number of shortest paths from s to t and $\sigma_{s,t}(e)$ the number of them that pass through edge e,

[40] Vladimir Batagelj and Andrej Mrvar. "A subquadratic triad census algorithm for large sparse networks with small maximum degree". In: *Soc. Networks* 23 (2001), pp. 237–243.

[41] M. Girvan and M. E. J. Newman. "Community structure in social and biological networks". In: *PNAS* (2002). DOI: `https://dx.doi.org/10.1073/pnas.122653799`. URL: `https://www.pnas.org/doi/full/10.1073/pnas.122653799`.

link betweenness of e in graph G is:

$$b(e, G) = \sum_{s,t \in V \,:\, s \neq t} \frac{\sigma_{s,t}(e)}{\sigma_{s,t}}$$

By removing the most central edge we are left with multiple connected components in G, they will represent our partition of V. We can also obtain a dendogram from this process by keeping track of which connected components are separated at each iteration. Once again we used the algorithm implementation provided by networkx in `networkx.algorithms.community.girvan_newman` that can be summed up as the pseudo-code in Listing 3.6.

```
def girvan_newman(G) -> Iterator[tuple[set[node]]]:
    # First iteration returns nodes in the original conn. components
    yield (set(cc.nodes) for cc in G.connected_components())
    g = G.to_undirected().remove_self_loops()
    while len(g.edges) > 0:
        n_communities = len(g.connected_components())
        new_communities = n_communities
        # Remove edges until we form a new conn. component
        while n_communities >= new_communities:
            # Find most central edge
            e = max(link_betweenness, g.edges)
            # Remove such edge
            g.remove_edge(e)
            # Check if removing the edge formed a new component
            new_communities = len(g.connected_components())
        # return the tuple of nodes in the connected components
        yield (set(cc.nodes) for cc in g.connected_components())
```

Listing 3.6: Girvan Newman algorithm

Another possible community algorithm is the **Louvain**'s algorithm, which is based on *modularity*. The idea behind the concept of **modularity** is that a community should contain more edges than expected in a random graph (from the Chung-Lu random graph model), so given a graph $G = (V, E)$ and a subset $S \subseteq V$ the *modularity* $M(S)$ is defined as:

$$M(S) = \frac{1}{2} \sum_{u,v \in S} \frac{deg(u)deg(v)}{2|E|}$$

The modularity of the partitioning of V in communities is the sum of the modularity of each community.

$$M(\mathcal{C}) = \sum_{C \in \mathcal{C}} M(C)$$

The Louvain's algorithm finds the partitioning with maximum modularity using a greedy agglomerative approach: it starts from an initial partitioning where each node is its own community, then merges the pair of communities C_i, C_j that maximize the modularity improvement when merged $\delta(\mathcal{C}, C_i, C_j) = M(\mathcal{C} - C_i - C_j + (C_i \cup C_j)) - M(\mathcal{C})$ (Listing 3.7). The algorithm continues iteratively until the best modularity improvement is below a given non-negative threshold. Listing 3.8 summarizes the networkx's implementation of the algorithm found in `networkx.algorithms.community.louvain_communities`.

```python
def compute_deltas(G, partitioning) -> dict[(int, int), float]:
    mod = modularity(G, partitioning)
    mod_deltas: dict[(int, int), float] = {}
    for i, c_i in enumerate(partitioning):
        for j, c_j in enumerate(partitioning[i:]):
            new_partitioning = partitioning.remove_at(i)
                                           .remove_at(j)
                                           .append(c_i + c_j)
            mod_deltas[(i, j)] = modularity(G, new_partitioning) - mod
    return mod_deltas
```

Listing 3.7: Python function to compute modularity difference of joining communities

```python
def louvain(G) -> list[set[node]]:
    partitioning = [{u} for u in G.nodes]
    while True:
        # Compute modularity differences
        mod_deltas = compute_improvements(G, partitionings)
        # Stop if there are no improving partitionings
        if max(mod_deltas.values()) <= 0:
            break
        # Update partitioning
        i, j = max(mod_deltas, key=mod_deltas.values())
        partitioning = partitioning.remove_at(i)
                                   .remove_at(j)
                                   .append(c_i + c_j)
    return partitioning
```

Listing 3.8: Louvain's greedy agglomeriative algorithm

The output of both community-finding algorithms is a mapping of nodes to the arbitrary index of the community they belong to, we could not use the community index as a feature by itself as it could change in value when computed on the same input graph so would just be noise. Instead, we create a binary feature `same_xx_community` to be added to settlement instruction features vector that is valued `1` when the two participants belong to the same community and `0` otherwise.

Table 3.7: Features from community algorithms

Feature	Description
	Whether the two participants of the instruction belong to the same community on the participants graph:
`same_gn_community`	when applying the Girvain-Newman algorithm.
`same_lv_community`	when applying the Louvain algorithm.

3.1.8 Node embeddings

The following approach is based on the concept of node *embedding*. The **embedding** z_v of a node v in a graph G is a point in R^d, where d is the dimensionality of the embedding space, obtained applying a function $f : V \to \mathbb{R}^d$ called *encoder*. The goal of the **encoder** is to learn a representation that preserves the similarities between nodes in the graph and their embeddings. The embeddings z_v of all nodes in the graph can be concatenated column-wise to obtain the Z matrix. The target dimensionality of the embedding space (d) is an input parameter of the embedding algorithms. The learning process is based on the **encoder-decoder** framework, which requires the definition of 4 components:

- Nodes similarity function $S(u, v)$
- Encoder function $f(u)$
- Embeddings similarity function $D(z_u, z_v)$
- Loss function $\ell(D(z_u, z_v), S(u, v)))$

One of the possible definitions of these components is given by **node2vec**[42] that is a **random-walk** approach. The idea behind random-walk approaches is that the similarity between nodes u and v is given by the probability of visiting v on a random walk of length k starting at node u, which we define as $P_k[v|u] = S(u, v)$. One property of such probability is that it can be easily and reliably approximated: consider computing s random walks of length k from node u, let $n_{RW,k,s}(v|u)$ be the number of such walks containing node v, then:

$$\mathbb{E}\left[\frac{n_{RW,k,s}(v|u)}{s}\right] = P_k[v|u]$$

In practice node2vec uses **biased random-walks** where the bias is based on two parameters p, q, where p determines the probability to go back to the previous node and q determines the ratio between "BFS" moves and "DFS" moves (see node2vec paper[42] for details). When the edges of the graphs are weighted the weights can also be used to regulate the bias of the next step. As embeddings similarity function $D(z_u, z_v)$ node2vec uses the negative sampling technique:

$$D(z_u, z_v) = log\left(\sigma(z_u^T z_v)\right) - \sum_{i=1}^{r} log\left(\sigma(z_u^T z_i)\right))$$

Where i is a node chosen from V with probability proportional to its degree and $\sigma(\cdot)$ is the sigmoid function. As loss function the random-walk approaches use the cross-entropy $H(p, q) = -\sum_{x \in \mathcal{X}} p(x)\, log\, q(x)$.

[42]Aditya Grover and Jure Leskovec. *node2vec: Scalable Feature Learning for Networks.* 2016. DOI: `10.48550/ARXIV.1607.00653`. URL: `https://arxiv.org/abs/1607.00653`.

Summarizing, the optimization problem that node2vec intends to solve is:

$$\boldsymbol{Z}^* = \underset{Z}{argmin} - \sum_{u\in V}\sum_{v\in V} \frac{n_{RW,k,s}(v|u)}{s} \left(log\left(\sigma(z_u^T z_v)\right) - \sum_{i=1}^{r} log\left(\sigma(z_u^T z_i)\right) \right)$$

and it tries to do so by using stochastic gradient descent.

The implementation of the node2vec algorithm we used is provided by Elior Cohen on Github[43] which easily integrates with the networkx's library. Using this algorithm we extracted a 8-dimensional feature vector (the embedding z) for each node in every graph, including the unweighted version of the algorithm, and the one biased using the amount and quantity edge weights (Table 3.8).

Table 3.8: Node2Vec feature vectors

Feature			Description
`n2v_0`	...	`n2v_7`	Unweighted embedding.
`n2v_amt_0`	...	`n2v_amt_7`	Embedding weighted on the amount.
`n2v_qty_0`	...	`n2v_qty_7`	Embedding weighted on the quantity.

3.2 Feature selection and source graph

At the end of the extraction process, we have more than 50 features to add to participants' data, we will now move on to address two issues that arise when using these features. The first issue is that the new feature vector we produce is even larger than the previously available features, and most of them look noisy and not informative. The second issue is related to chapter 2, in particular, we have to determine what partitioning choice to apply to produce the **source graphs** to extract the features.

To solve the first issue we computed the features on the original MultiDiGraph, then applied feature selection using a few different scores. The first score we tried is (Pearson's) correlation coefficient[44], which measures the linear correlation between two sets of data. To compute the correlation we used the Pandas' `corr` method applied to all the features against the target variable (`cd_sett`). Unfortunately, the score of graph-extracted features has a very low value, less than the absolute value

[43]Elior Cohen. *Node2Vec - Github.* URL: `https://github.com/eliorc/node2vec`.

[44]Wikipedia. *Pearson correlation coefficient.* URL: `https://en.wikipedia.org/wiki/Pearson_correlation_coefficient`.

of `0.06`, much lower than other already-available features. Although disappointing this only proved that there is no evidence of a direct linear correlation between the features and the target variable, but there could be more complicated relationships so we moved on to the next score. A second score is based on training an explainable[45] machine learning model and observing how much they weigh each input feature. The model we chose for this experiment is the widely-used XGBoost[46] (as seen in chapter 1.6.1), which provides three metrics of "feature importance": weight, gain and cover. We won't dwell on the details of these three metrics, but intuitively the higher the average of the three values, the more important the variable. The results in figure 3.4 show that the model uses some of the added features to make the prediction. One last score that we will only marginally cite is the **predictive**

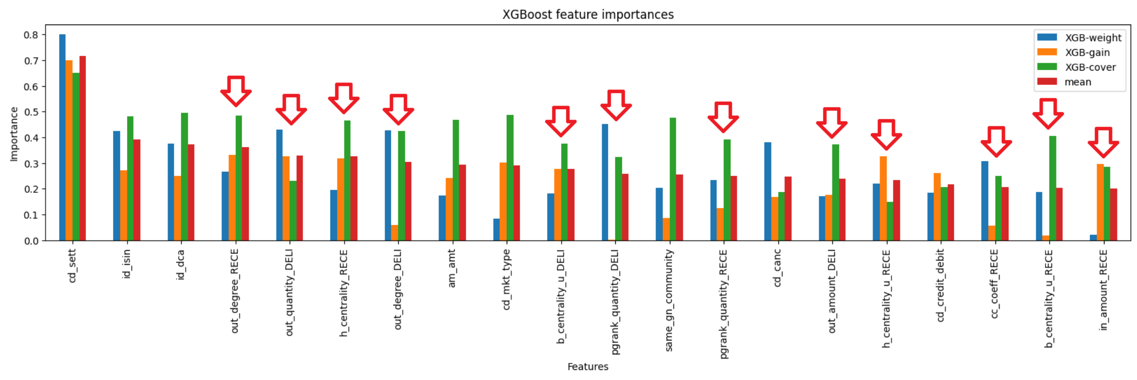


Figure 3.4: XGBoost feature importances

power score (or pp-score), which is able to detect non-linear relationships between a feature and a target variable. In brief, the pp-score trains a ML model on a single feature, then it computes its F1 and normalizes it over the F1 score of the naive model that always predicts the most frequent/average value. Results showed that the pp-score of some of the new features is not significantly higher than most of the already-available features (Figure 3.5).

From the previous analyses, we selected a subset of the available graph-extracted features to be used in the next steps; we choose the features with the following criteria: they have to have an appreciable amount of variance per participant, they must have been weighted by XGBoost in at least one training and have a relatively

[45]Wikipedia. *Explainable AI.* URL: `https://en.wikipedia.org/wiki/Explainable_artificial_intelligence`.

[46]Github Distributed Machine Learning Community. *XGBoost.* URL: `https://github.com/dmlc/xgboost`.

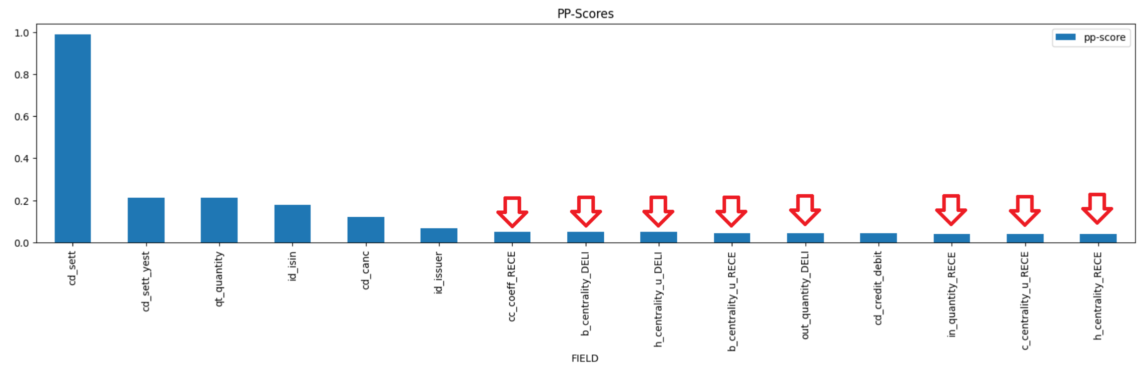


Figure 3.5: PP-Score of features

high pp-score. The resulting list of 16 features is described in Table 3.9

Table 3.9: Graph-features used in next steps

`degree`	`out_degree`	`in_degree`	`in_amount`
`out_amount`	`in_quantity`	`out_quantity`	`cc_coeff`
`c_centrality`	`c_centrality_u`	`h_centrality`	`h_centrality_u`
`b_centrality`	`b_centrality_u`	`pgrank_amount`	`pgrank_quantity`

As one can see from the table, triads, communities and node embeddings were discarded, although since our first attempts we optimized the handling of features so we are now able to use more of them, thus we will probably consider them in future analyses.

Now that we have reduced the number of features and kept only the most meaningful ones, we can proceed to solve the second issue: the choice of the type of graph to extract features from. To tackle this second issue we performed a "grid search": we tried two of the possible node identifiers, SAC and BIC, with all three possible instructions subsets (complete, failed instructions only, grouped by security), for a total of six possible features "source" graphs (table 3.10).

We will now review how we modified the current approach to include the graph features. The first step was to create the nine graphs, extract the node-level features and store them for later use. One day of instructions data is loaded in a pandas DataFrame from S3, a filtering function is applied based on the chosen subset, and the DataFrame is converted into MultiDiGraph graph using `nx.from_pandas_edgelist`. The MultiDiGraph G is converted to a simple undirected graph uG and stored, it will be used to compute metrics on the undirected version of the graph. A custom

Identifier	Subset	Description
BIC	Complete	All instructions, BIC as node identifier
	Failed	Only previously-failed instructions, BIC as node identifier
	ISIN	One graph per security, BIC as node identifier
SAC	Complete	All instructions, SAC as node identifier
	Failed	Only previously-failed instructions, SAC as node identifier
	ISIN	One graph per security, SAC as node identifier

Table 3.10: Source graphs

python module we developed takes as input the two graphs and a list of metrics to compute, it applies the previously seen algorithms for the chosen metrics and it returns them as a DataFrame indexed by node identifier (SAC or BIC). When completed, each daily DataFrame is uploaded to S3 as a partition of a parquet file.

When the metrics for all dates are computed, we can use them in the preprocessing phase of the machine learning pipeline. During preprocessing the "raw" data is loaded in the SageMaker instance to be "cleaned", to extract new features from and to be encoded, ready to be fed to a machine learning model. Together with raw data we also load the graph metrics; to merge them together we use the pandas DataFrame "join" method that, similarly to SQL joins, combines fields of one table with fields from another table when the value on one or more common fields matches. In our case we join the graph features table twice, once for the deliverer party and once for the receiver party, so the new feature column names will be in the form of `[feature]_DELI` and `[feature]_RECE`; which columns to use in the join of the two datasets depends on the chosen node identifier of the graph features. Note that the relationship between a row of the raw dataset and the graph features dataset is many-to-two (one for the deliverer and one for the receiver, unless they coincide). Listing 3.9 shows how to perform the join.

```python
# Rename columns as {c}_PTY
deli_gfeat = gfeat_df.rename(columns={c: f"{c}_DELI" for c in
    gfeatures_df.columns if c not in ["date", "node"]})

# Append party (one) features
pd.merge(
    left=raw_df, right=pty_gfeat,
    left_on=("id_deli_sac_t2s", "dt_business"),  right_on=("node_id",
    "date"),
    how="left", validate="m:1"
)

# Similarly to rece party
...
```

Listing 3.9: Join raw data and graph-extracted data

An alternative possible way to join the two datasets would be to use Glue Spark jobs: one could load the two datasets from S3 in Spark Dataframes and perform the join in SparkSQL for better performance. In ESM a different machine-learning model is trained for each client, this requires that the training dataset for each model only contains instructions regarding that client. For this reason the preprocessed dataset is split by participant and stored in S3 to be used to train models in SageMaker training jobs. A custom module launches a training job for each combination of hyperparameters in a grid search fashion; for our experiments we used the same hyperparameters as the already-in-use models to have a fairer comparison. When the training jobs are completed, they get compared and the best 5 models are retained while the others are deleted. Although the models we train are binary classificators over the instructions settlement state, the evaluation metrics are based on the final task of regression (prediction of the imbalance):

- **M**ean **A**bsolute **E**rror: $\frac{1}{n}\sum_{i=1}^{n}|Y_i - \hat{Y}_i|$
- **R**oot **M**ean **S**quared **E**rror: $\frac{1}{n}\sqrt{\sum_{i=1}^{n}\left(Y_i - \hat{Y}_i\right)^2}$

This difference in evaluation is due to the optimization process already in place: because the final task is the imbalance prediction, the best classification models are chosen based on the counter value of the correctly classified instructions so that the process may choose models with lower accuracy than the best available ones. In this approach, we are able to optimize for imbalance prediction because a model only considers client-related instructions and it is easy and clear how to compute the regression.

3.3 Results

With the previously described setup we extracted features and trained models for three ESM clients over the same raw data source, they were chosen as we have custom and better-performing prediction models to compare our models to. The new "graph-based" models will be compared with two "baseline" models, the naïve model which predicts 100% of the instructions as settled, and the production models which are the prediction models that are currently providing predictions for each client.

To better contextualize the results we provide some settlement statistics about clients in table 3.11; note that the clients' names were anonymized for privacy.

Table 3.11: Clients statistics

Client	Daily imbalance			% instr. under avg. amount
	mean ± SD.	min.	max.	
Client 1	3.85e6 € ± 74.6e6 €	-0.99e9 €	1.0e9 €	97.5%
Client 2	2.89e7 € ± 11.7e7 €	-54.0e7 €	72.0e7 €	94.8%
Client 3	1.10e9 € ± 2.13e9 €	-10.7e9 €	9.23e9 €	86.3%

Continuing to the results of the approach, tables 3.14, 3.13, and 3.12 report the best evaluation metrics of trainings for each combination of input graph-extracted features, each followed by a line chart of three models accuracies of imbalance prediction over the validation and test set: the naive model (naïve), the current production model (prod), and the best-performing graph-based model from the respective client's table (see figure's caption).

Table 3.12: Client 1

Identifier	**Subset**	MAE	RMSE	Δ MAE
naïve		24.7e6	4.3e7	+17.7e6
prod		7.0e6	1.6e7	-
BIC	Complete	5.5e6	1.3e7	-1.5e6
BIC	Failed	5.6e6	1.1e7	-1.4e6
BIC	**ISIN**	**2.4e6**	**4.4e6**	**-4.6e6**
SAC	Complete	7.2e6	1.4e7	+0.2e6
SAC	Failed	5.4e6	8.8e6	-1.5e6
SAC	ISIN	6.5e6	1.2e7	-0.5e6

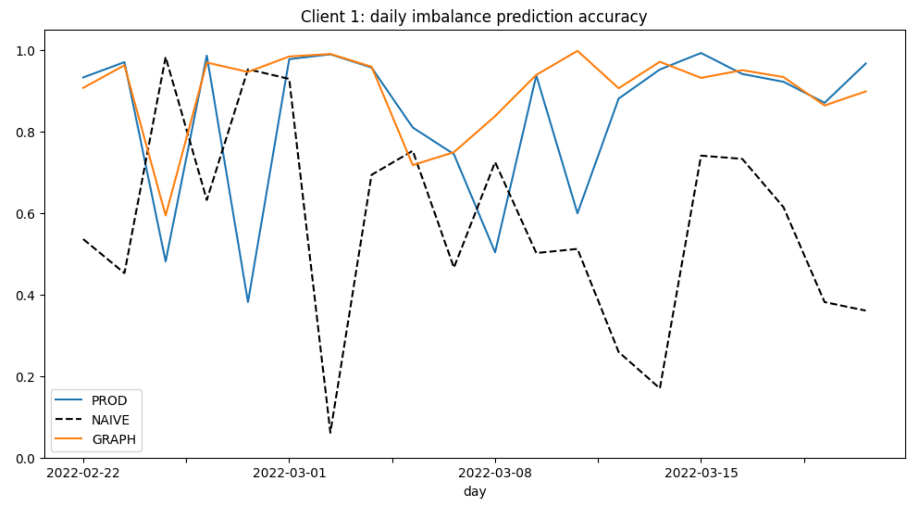


Figure 3.6: Client 1: Imbalance prediction accuracy of models (higher is better) "GRAPH" is (BIC, Failed)

Table 3.13: Client 2

Identifier	**Subset**	MAE	RMSE	Δ MAE
naïve		13.6e7	16.3e	+7.8e7
prod		5.8e7	12.4e7	-
BIC	**Complete**	**4.3e7**	**5.9e7**	**-1.5e7**
BIC	Failed	5.1e7	7.2e7	-0.7e7
BIC	ISIN	4.4e7	6.9e7	-1.4e7
SAC	Complete	5.1e7	9.5e7	-0.7e7

Continued on next page

Table 3.13: Client 2 (Continued)

SAC	Failed	4.9e7	8.2e7	-0.9e7
SAC	ISIN	4.8e7	6.8e7	-1.0e7

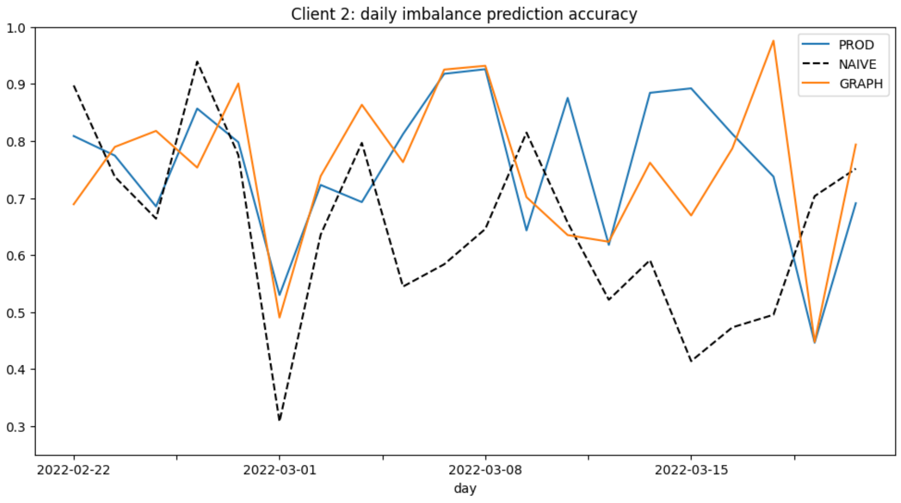


Figure 3.7: Client 2: Imbalance prediction accuracy of models (higher is better) "GRAPH" is (BIC, Complete)

Table 3.14: Client 3

Identifier	**Subset**	MAE	RMSE	Δ MAE
naïve		12.2e8	10.6e8	+7.5e8
prod		**4.7e8**	**5.7e8**	-
BIC	Complete	5.7e8	7.3e8	+1.0e8
BIC	Failed	5.5e8	8.4e8	+0.8e8
BIC	ISIN	5.2e8	6.7e8	+0.5e8
SAC	Complete		failed[1]	
SAC	Failed		failed[1]	
SAC	ISIN		failed[1]	

[1] Instance kept shutting down due to full memory, we were not able optimize the training process in time.

The tables show that results vary between clients. Both client 1 and client 2 have models with consistently better results than the current production models; client 1 also has one outstanding model with 35% of the production error. Unfortunately, client 3 shows no sign of improvement over multiple tests, this might be

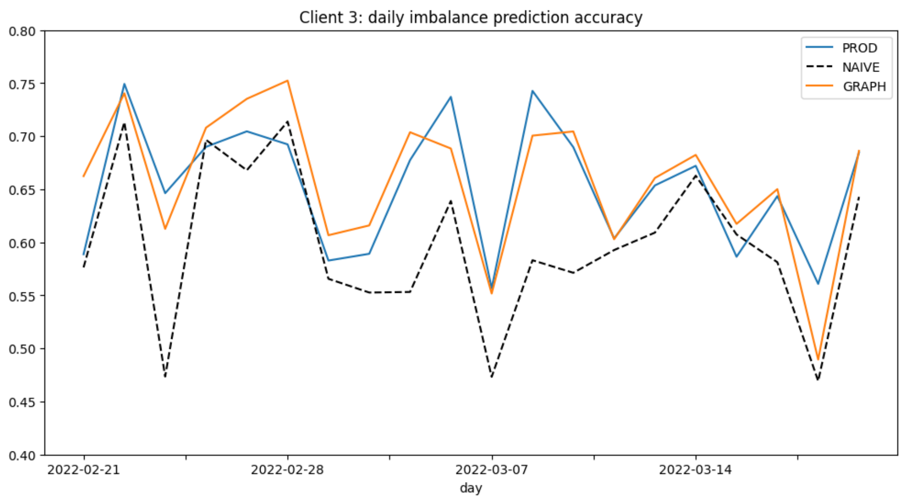


Figure 3.8: Client 3: Imbalance prediction accuracy of models (higher is better) "GRAPH" is (BIC, ISIN)

due to the client having few but large instructions which means that little changes in the classification accuracy lead to large forecasting errors. When using features extracted from source graphs built using SAC identifier the training kept failing for full memory, this is still under investigation.

Chapter 4

Approach 2: GNNs

In this last chapter we will introduce another approach we employed to extract information from graphs which is also based on the **traditional framework** we have seen earlier.

4.1 Introduction to GNNs

Graph neural networks (GNNs) are neural networks whose structure changes based on the input graph; they take as input a graph $G = (V, E)$, where vertices have an associated feature vector x_v, and apply a layer-styled processing to each x_v using the data from neighboring vertices $x_i : i \in \mathcal{N}(v)$. The problem that graph neural networks solve is *node embeddings* that we have seen in chapter 3.1.8, which is to find a way to represent a node $v \in G$ as a d-dimensional vector z_v such that its similarity to other nodes is preserved. One of the advantages of using a graph neural network to produce embeddings is that we can incorporate node features in the computation of the embedding to obtain better representations.

4.1.1 Message passing framework

But how could we include graph information in a neural network computation? A naïve approach would be to concatenate the adjacency matrix vector A_v of node v to its feature vector x_V and feed it to a multi-layered neural network. Unfortunately, this approach would not work as it depends on the arbitrary ordering of the nodes (not *permutation equivariant*) and would not be applicable on new graphs if they were to differ in the number of nodes.

To solve these problems, graph neural networks use the **message passing framework**. In the message passing framework the computation proceeds in iterations, at iteration k the "intermediate" embedding of node v is $h_v^{(k)}$ and it is computed using its previous embedding and the embeddings of its neighbors $u \in \mathcal{N}$. At each iteration (also called layer) two functions are applied for each node of the graph: $\texttt{AGGREGATE}^{(k)}\left(h_u^{(k)}, \forall u \in \mathcal{N}(v)\right)$ which produces message $m_{\mathcal{N}(v)}^{(k)}$ and $\texttt{UPDATE}^{(k)}\left(h_v^{(k)}, m_{\mathcal{N}(v)}^{(k)}\right)$. In general, these two functions can be any differentiable function to produce a functioning neural network. The AGGREGATE function takes as input the intermediate embedding of the neighbors of v and produces a vector we call the message of iteration k. The UPDATE function takes the previous intermediate embedding of node v and the aggregation message to produce the embedding of iteration $k + 1$. For the GNN to be *permutation equivariant* these two function must be "proper" functions working on unordered sets. Figure 4.1 contains an example reference graph that is used in figure 4.2 to show two layers of message passing framework's computation for node A.

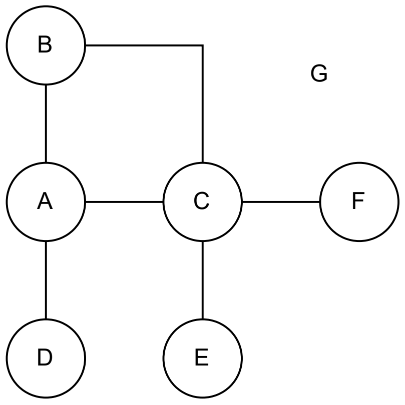


Figure 4.1: Example graph

A GNN "family" is defined by the AGGREGATE and the UPDATE functions, we will now see three examples: the most basic version of GNN, graph convolutional networks and graph SAGE.

4.1.2 Basic GNN

The most basic version of a GNN is given by:

- $m_{\mathcal{N}(v)}^{(k)} = \texttt{AGGREGATE}^{(k)}(\{h_u^{(k)}, \forall u \in \mathcal{N}(v)\}) = \sum_{u \in \mathcal{N}(v)} h_u^{(k)}$
- $h_v^{(k+1)} = \texttt{UPDATE}^{(k)}(h_v^{(k)}, m_{\mathcal{N}(v)}^{(k)}) = \sigma\left(W_{\text{self}}^{(k+1)} h_v^{(k)} + W_{\text{neigh}}^{(k+1)} m_{\mathcal{N}(v)}^{(k)} + b^{(k+1)}\right)$

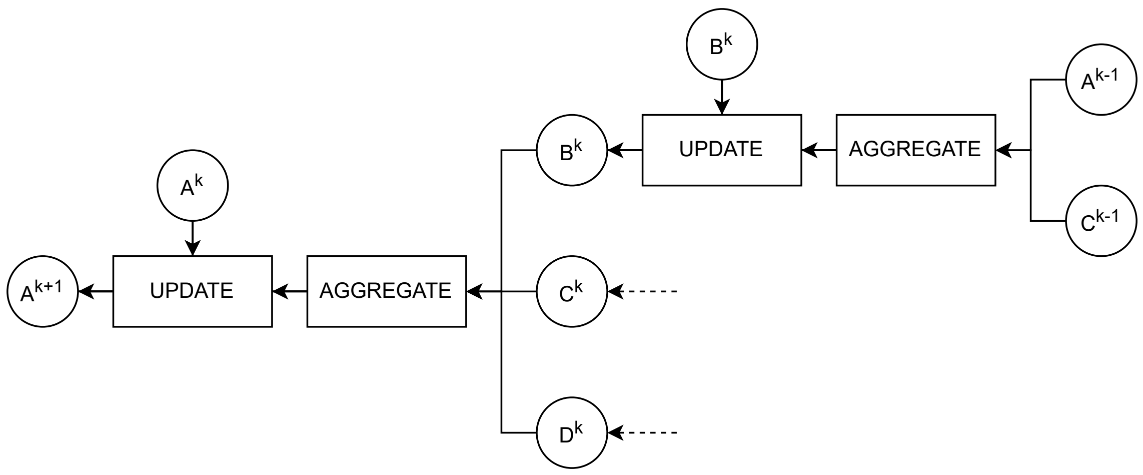


Figure 4.2: Message passing framework computation

Where $W_{\text{self}}^{(k+1)}$ and $W_{\text{neigh}}^{(k+1)}$ are trainable weight matrices of iteration k, with $W_{\text{self}}^{(k+1)}, W_{\text{neigh}}^{(k+1)} \in \mathbb{R}^{d^{(k+1)} \times d^{(k)}}$, $b^{(k+1)}$ is the bias, and $\sigma(\cdot)$ is an elementwise non-linear function (e.g. ReLU). Putting it all together, for each node v at each iteration:

$$h_v^{(k+1)} = \sigma\left(W_{\text{self}}^{(k+1)} h_v^{(k)} + W_{\text{neigh}}^{(k+1)} \sum_{u \in \mathcal{N}(v)} h_u^{(k)} + b^{(k+1)}\right)$$

We can also define the basic GNN using graph-level equations:

$$H^{(k+1)} = \sigma\left(H^{(k)} W_{\text{self}}^{(k+1)} + A H^{(k)} W_{\text{neigh}}^{(k+1)} + B\right)$$

where A is the adjacency matrix of G, $H^{(k)}$ is the matrix where each row is the embedding of a node at layer k, and B is the bias matrix where each row is b. The graph-level equation shows that GNNs can be efficiently implemented with a small number of matrix operations.

We can reduce the number of parameters (and the complexity of the model) by **sharing parameters** across iterations, meaning that $W_{\text{self}}^{(k+1)} = W_{\text{self}}$, $W_{\text{neigh}}^{(k+1)} = W_{\text{neigh}}$, and $b^{(k+1)} = b$ for all k.

4.1.3 Graph Convolutional Networks GCNs

Another simplification that can be applied to the basic GNN is introducing **self-loops**: the previous intermediate embedding $h(k)_v$ of node v is considered in the `AGGREGATE`$^{(k)}$ function and the `UPDATE`$^{(k)}$ function only takes the message as input

$$h_v^{(k+1)} = \sigma\left(W^{(k+1)} \sum_{u \in \mathcal{N}(v) \cup \{v\}} h_u^{(k)} + b^{(k+1)}\right)$$

An issue of the basic GNN is the high sensitivity to node degrees, to solve this problem we can normalize the `AGGREGATE` function taking into account the node degree with a process called **neighborhood normalization**

$$m_{\mathcal{N}(v)}^{(k)} = \sum_{v \in \mathcal{N}(u)} \frac{h_v^{(k)}}{\sqrt{|\mathcal{N}(u)||\mathcal{N}(v)|}}$$

Introduced by Kipf and Welling in "Semi-supervised classification with graph convolutional networks"[47], graph convolutional networks are similar to basic GNNs but they use *self-loops* and *neighborhood normalization.*

4.1.4 Graph SAGE

The previous two families of GNNs both used the sum of embeddings as `AGGREGATE` operator, Graph SAGE[48] was the first to generalize the `AGGREGATE` function and use concatenation between the previous embedding and the message in the `UPDATE` function.

$$h_v^{(k+1)} = \sigma\left(W_{\text{self}}^{(k+1)} h_v^{(k)} \cdot W_{\text{neigh}}^{(k+1)} m_{\mathcal{N}(v)}^{(k)}\right)$$

Common choices for the `AGGREGATE` function are:

- mean of $h_v^{(k)}$: $m_{\mathcal{N}(v)}^{(k)} = \sum_{u \in \mathcal{N}(v)} \frac{h_u^{(k)}}{|\mathcal{N}(v)|}$
- element-wise max pooling: $m_{\mathcal{N}(v)}^{(k)} = max\left(\{\sigma\left(W_{\text{pool}}^{(k)} h_u^{(k)} + b\right), \forall u \in \mathcal{N}(v)\}\right)$
- first apply a neural network to each $h_v^{(k)}$ with learnable parameters θ and then apply an element-wise operator γ such as mean or max:
  $m_{\mathcal{N}(v)}^{(k)} = \gamma\left(\{\text{NN}_\theta(h_u^{(k)}), \forall u \in \mathcal{N}(v)\}\right)$

4.2 Usage

In chapter 3 we have seen the traditional framework for machine learning with graphs (figure 3.2), and we have chosen a list of hand-picked features to carry some

[47]Thomas N. Kipf and Max Welling. “Semi-Supervised Classification with Graph Convolutional Networks”. In: *CoRR* abs/1609.02907 (2016). arXiv: `1609.02907`. URL: `http://arxiv.org/abs/1609.02907`.

[48]William L. Hamilton, Rex Ying, and Jure Leskovec. “Inductive Representation Learning on Large Graphs”. In: *CoRR* abs/1706.02216 (2017). arXiv: `1706.02216`. URL: `http://arxiv.org/abs/1706.02216`.

of the information contained in graphs to a machine learning model. With GNNs we want to skip the process of selection and computation of node-level features, and let a training algorithm determine the best way to transform (embed) a node into a vector that can be fed to a machine-learning model. GNNs are capable of embedding nodes but not edges, although there are some possible workarounds we deemed not suitable for our purposes; for this reason in GNNs we will only use *instruction-centric* graphs that we have seen in chapter 2.2, where nodes represent the instructions that we will classify. For this task we chose to use graphs built with all the instructions from one day, using the SAC as node identifier. Contrary to the previous approach, we will not develop a different prediction model for each of the clients by filtering the input instructions to build the graph because we would lose information about the rest of the settlement environment. Now that we have chosen the graph modeling for the input data we are only left with the task of defining a suitable model.

4.3 Models

We have seen that GNNs proceed in iteration/layers, we can mix these layers with other typical neural network layers to obtain better results. To define the model we will start from the current prediction model: a categorical embedding layer followed by a few dense layers and one last neuron with sigmoid activation function for binary classification, for more details see Maria Giuseppina Brunelli's thesis[29]. We edited this model by adding one highly-dimensional dense layer after the categorical embedding and two graph neural network layers between that and the few dense layers, figure 4.3 shows the template structure of our GNN-based model. The choice of the number of GNN layers for our model is to avoid *oversmoothing*. **Oversmoothing** is a graph neural networks issue where if the number of layers is too high the final embedding of a node will depend on many nodes, leading to a similar embedding for all nodes. Because our graphs have a small diameter, we have to choose a small number of GNN layers (two) to prevent most embeddings to be based on the majority of the graph.

To implement the model we used TensorFlow together with Python DGL li-

[29]Maria Giuseppina Brunelli. *Cash flow forecasting during night-time settlement cycle as a way to improve settlement efficiency in target2-securities.* 2021.

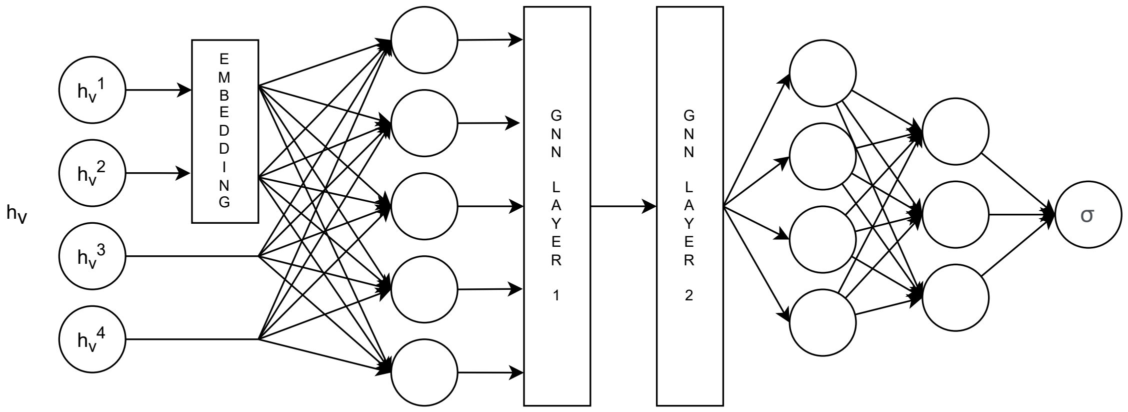


Figure 4.3: Template structure of our GNN-based models

brary[49] for graph neural networks, which has the advantage of easily integrating with networkx. DGL provides different implementations of GNN layers in TensorFlow, among which we used `GraphConv` for graph convolutional networks and `SAGEConv` for graph SAGE. The training of the model is performed in batches of graphs of (maximum) size of 10.000 vertices, because, as explained in 2, it has reasonable memory size and conversion-from-instructions time. The following models are trained

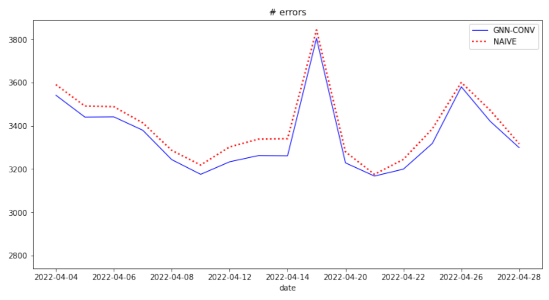
4.4 Evaluation and results

Contrarily to the previous approach, we evaluated the model using binary classification metrics such as accuracy, F1-score and precision-recall AUC. Given that the model takes all instructions from a day as input, we want it to perform better than the naive classification model; optionally we would also want it to have comparable performances to the current production models that are specialized in single clients.

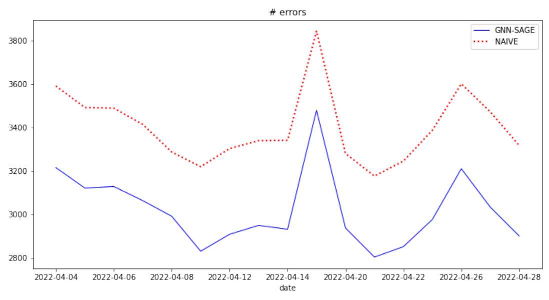
We tested the models using settlement instructions from April 2022, and the following figures show the comparison in the number of incorrectly classified settlement instructions per day per model. From the comparison (figure 4.4) one can clearly see that the SAGE-based model performs much better than the GCN-based one as it makes much less errors than the naive model; this intuition is confirmed by the evaluation metrics in table 4.1

We also compared the two models to the current three company-related production models; these three models were trained with instructions until 31-03-22 and we compared predictions for April 2022.

[49] *Deep Graph Library*. URL: `https://www.dgl.ai/`.



(a) GraphConv



(b) SAGEConv

Figure 4.4: Comparison of the number of errors between GNN-like models and the naïve over all instructions from April

Model	**Accuracy**	**F1-score**	**PR-AUC**
Naïve	93.05 %	96.40 %	95.52 %
GCN-based	94.25 %	96.41 %	96.64 %
SAGE-based	**95.13** %	**96.84** %	**97.18** %

Table 4.1: GNN-based models evaluation metrics

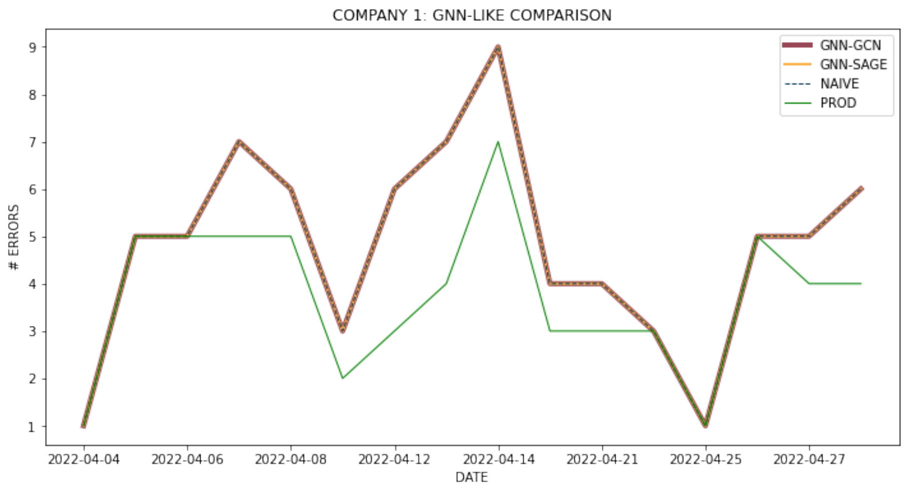


Figure 4.5: Client 1: models comparison in the number of errors (lower is better)

Model	**Accuracy**	**MAE**	**RMSE**	**Δ MAE**
Naïve	98.2 %	2.9e6	4.1e6	+2.0e6
Prod	**98.6** %	**8.7e5**	**1.8e6**	-
GCN-based	98.2 %	2.9e6	4.1e6	+2.0e6
SAGE-based	98.2 %	2.9e6	4.1e6	+2.0e6

Table 4.2: Client 1: Prod and GNN-based models evaluation metrics

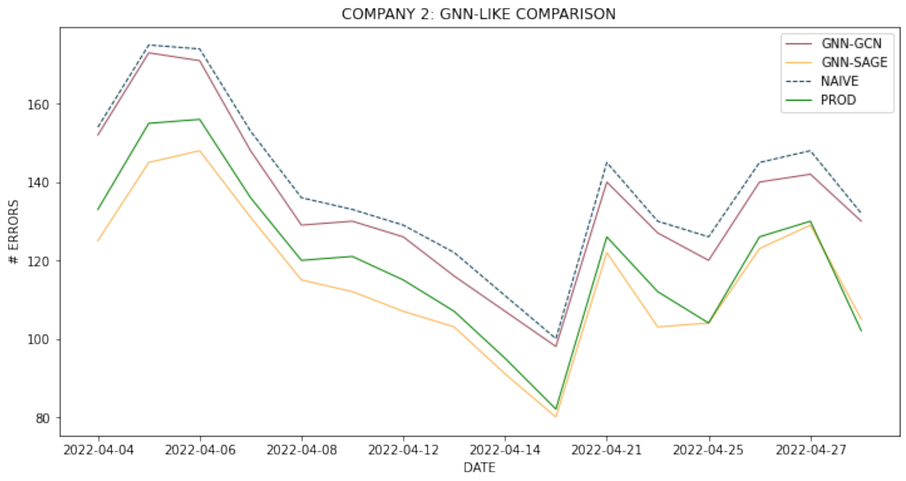


Figure 4.6: Client 2: models comparison in the number of errors (lower is better)

Model	**Accuracy**	**MAE**	**RMSE**	**Δ MAE**
Naïve	96.4 %	42.2e6	78.2e6	0
Prod	96.9 %	42.2e6	78.2e6	-
GCN-based	96.8 %	42.2e6	78.2e6	0
SAGE-based	**97.2** %	**41.8e6**	78.2e6	**-0.4e6**

Table 4.3: Client 2: Prod and GNN-based models evaluation metrics

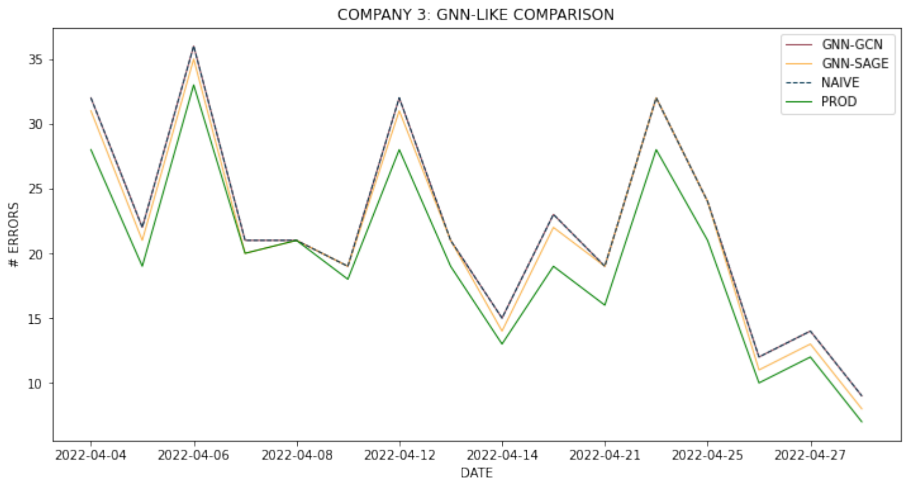


Figure 4.7: Client 3: models comparison in the number of errors (lower is better)

Model	**Accuracy**	**MAE**	**RMSE**	**Δ MAE**
Naïve	92.3 %	1.34e9	1.48e9	+5.9e6
Prod	**93.2** %	1.34e9	1.48e9	-
GCN-based	92.3 %	1.34e9	1.48e9	+5.8e6
SAGE-based	92.4 %	1.34e9	1.48e9	+5.3e6

Table 4.4: Client 3: Prod and GNN-based models evaluation metrics

The tables show that the results vary between clients. Only customer 2 has the SAGE-based model with consistently better results than current production models; client 1 and client 3 did not benefit from this technique. The explanation for this behavior could lie in the fact that client 2 has a greater focus on equities, which are more affected by the effects of daisy chains and volatility than the other two clients who manage mainly government bonds.

Chapter 5

Conclusions and future works

In this last section we will review the topics covered and the results obtained from our research. We applied two approaches to employ relationships information to improve the prediction of settlement status of settlement instructions, both based on graphs; the first approach was to extract node-level metrics as feature vectors to concatenate to existing features, and the second one was to introduce graph neural network layers to the machine learning model. Before applying the information extraction approaches we have also seen how to transform a list of settlement instructions into participant-centric graphs and instruction-centric graphs, by using different node identifiers and partitioning choices. For the first approach we have seen the list of node-level metrics we have extracted and the respective algorithms we used to compute them; as for results we showed their feature importance and their impact on the predictions, which was positive for two of ESM clients, but negative for Client 1. For the second approach we have reviewed the foundations of graph neural networks and the message-passing framework that they are based on, we have seen how we introduced two kinds of graph neural network layers in our prediction model and the results, showing that the SAGE-based model has much better performances than the naive model in prediction over all instructions, but it is not always better when compared to already-in-production client-specific models.

This thesis presented a first rough approach to graph information extraction for the settlement status prediction problem, leaving considerable space for future, more insightful work. For the topic of graph features extraction (chapter 3) we could

- include features from different source graphs at once
- compute more node-level metrics (e.g. motifs)

- include features from previous days (t-1 and t-2)
- compute and include graph-level metrics (e.g. global clustering coefficient)
- optimize or parallelize metrics computation to speed up the computation

As for graph neural networks (chapter 4) there are some points that can be explored

- different choices of `UPDATE` and `AGGREGATE` functions (e.g. Graph Isomorphism Networks, Graph Attention Networks)
- input graphs using different node identifiers (before the computation of the line graph) and additional subsets (section 2.3.2)
- develop a different model for each client by keeping the same input and applying a filter to the samples to be considered in the loss function based on the selected client

Bibliography

[1] Michael Simmons. *Securities operations.* en. The Wiley Finance Series. Nashville, TN: John Wiley & Sons, Mar. 2002.

[2] Investopedia. *Financial Asset Definition.* URL: https://www.investopedia.com/terms/f/financialasset.asp.

[3] European Central Bank (ECB). *Glossary of terms related to Payment, Clearing and Settlement systems.* URL: https://www.ecb.europa.eu/pub/pdf/other/glossaryrelatedtopaymentclearingandsettlementsystemsen.pdf.

[4] Investopedia. *Stock.* URL: https://www.investopedia.com/terms/c/stock.asp.

[5] Investopedia. *Ordinary dividend.* URL: https://www.investopedia.com/terms/c/ordinary-dividends.asp.

[6] Investopedia. *Corporate Action.* URL: https://www.investopedia.com/terms/c/corporateaction.asp.

[7] *Euronext Securities Milan.* URL: https://www.euronext.com/en/post-trade/euronext-securities/milan.

[8] Banca d'Italia. *Central counterparty.* URL: https://www.bancaditalia.it/compiti/sispaga-mercati/controparte-centrale/index.html.

[9] ESM. *Bilateral netting report - Linkedin.* URL: https://www.linkedin.com/feed/update/urn:li:activity:6960205218395402241/.

[10] Clearstream. *Clearstream Banking's TARGET2-Securities Glossary.* 2022. URL: https://www.clearstream.com/resource/blob/1316800/461f1a121cd02eccac538e760de29042/t2s-glossary-data.pdf.

[11] *T2S - Matching fields from a message perspective.* 2015. URL: https://www.ecb.europa.eu/paym/target/t2s/profuse/shared/pdf/insights_on_matching_fields_from_a_message_perspective_t2s.pdf.

[12] François Villeroy de Galhau. *Payments and market infrastructures in the digital era.* Banque de France, 2018. URL: https://publications.banque-france.fr/sites/default/files/media/2021/01/07/payments_market.pdf.

[13] *TARGET2-Securities.* URL: https://www.ecb.europa.eu/paym/target/t2s.

[14] European Central Bank (ECB). "Settlement fails – report on securities settlement systems". In: (2011). URL: https://www.ecb.europa.eu/pub/pdf/other/settlementfails042011en.pdf.

[15] Investopedia. *Naked Short Selling.* URL: https://www.investopedia.com/terms/n/nakedshorting.asp.

[16] *T2S Penalty Mechanism.* 2017. URL: https://ecsda.eu/wp-content/uploads/2019/01/Annex_I_1_T2S_Penalty_Mechanism.pdf.

[17] European Securities and Markets Authority. "ESMA Report on Trends, Risks and Vulnerabilities - 2020". In: (2020). URL: https://www.esma.europa.eu/sites/default/files/library/esma_50-165-1287_report_on_trends_risks_and_vulnerabilities_no.2_2020.pdf.

[18] European Securities and Markets Authority. "ESMA Report on Trends, Risks and Vulnerabilities - 2022". In: (2022). URL: https://www.esma.europa.eu/sites/default/files/library/esma50-165-2061_trv_1-22_statistical_annex.pdf.

[19] Wikipedia. *Binary classification.* URL: https://en.wikipedia.org/wiki/Binary_classification.

[20] Rehmsmeier M Saito T. "The Precision-Recall Plot Is More Informative than the ROC Plot When Evaluating Binary Classifiers on Imbalanced Datasets". In: (2015). URL: https://doi.org/10.1371/journal.pone.0118432.

[21] AWS. *S3.* URL: https://aws.amazon.com/s3/.

[22] Apache. *Parquet.* URL: https://parquet.apache.org/.

[23] Tianqi Chen and Carlos Guestrin. "XGBoost". In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining.* ACM, Aug. 2016. DOI: 10.1145/2939672.2939785. URL: https://doi.org/10.1145%2F2939672.2939785.

[24] Deloitte. *Artificial intelligence in post-trade processing.* URL: https://www2.deloitte.com/content/dam/Deloitte/us/Documents/technology/us-artificial-intelligence-in-post-trade-processing.pdf.

[25] Clearstream. *Xact Web Portal.* URL: https://www.clearstream.com/clearstream-en/products-and-services/connectivity-1-/clearstreamxact/xactwebportal/xact-web-portal-settlement-tutorials-3099658.

[26] SettlementDrive. *EasyFocus.* URL: https://www.euroclear.com/newsandinsights/en/Format/Webinars/liquidity-drive-and-settlement-drive-am.html.

[27] Wikipedia. *Autoregressive integrated moving average model.* URL: https://en.wikipedia.org/wiki/Autoregressive_integrated_moving_average.

[28] Wikipedia. *Autoregressive moving average model.* URL: https://en.wikipedia.org/wiki/Autoregressive%E2%80%93moving-average_model.

[29] Maria Giuseppina Brunelli. *Cash flow forecasting during night-time settlement cycle as a way to improve settlement efficiency in target2-securities.* 2021.

[30] AWS. *Sagemaker.* URL: https://aws.amazon.com/sagemaker/.

[31] *Pandas Python library.* URL: https://pandas.pydata.org/.

[32] Wikipedia. *Line graph.* URL: https://en.wikipedia.org/wiki/Line_graph.

[33] S. Wasserman and K. Faust. *Social Network Analysis: Methods and Applications.* pg. 201. 1994.

[34] David Eppstein and Joseph Wang. "Fast Approximation of Centrality". In: (2000). DOI: 10.48550/ARXIV.CS/0009005. URL: https://arxiv.org/abs/cs/0009005.

[35] Networkx. *Closeness centrality.* URL: https://networkx.org/documentation/stable/reference/algorithms/generated/networkx.algorithms.centrality.closeness_centrality.html.

[36] Networkx. *Betweenness centrality.* URL: https://networkx.org/documentation/stable/reference/algorithms/generated/networkx.algorithms.centrality.betweenness_centrality.html.

[37] Lawrence Page et al. *The PageRank Citation Ranking: Bringing Order to the Web.* Technical Report 1999-66. Previous number = SIDL-WP-1999-0120. Stanford InfoLab, Nov. 1999. URL: http://ilpubs.stanford.edu:8090/422/.

[38] Networkx. *Pagerank coefficient.* URL: https://networkx.org/documentation/stable/_modules/networkx/algorithms/link_analysis/pagerank_alg.html.

[39] Faust Wasserman. “Social Network Analysis: Methods and Applications”. In: (1994). DOI: https://doi.org/10.1017/CBO9780511815478.

[40] Vladimir Batagelj and Andrej Mrvar. “A subquadratic triad census algorithm for large sparse networks with small maximum degree”. In: *Soc. Networks* 23 (2001), pp. 237–243.

[41] M. Girvan and M. E. J. Newman. “Community structure in social and biological networks”. In: *PNAS* (2002). DOI: https://dx.doi.org/10.1073/pnas.122653799. URL: https://www.pnas.org/doi/full/10.1073/pnas.122653799.

[42] Aditya Grover and Jure Leskovec. *node2vec: Scalable Feature Learning for Networks.* 2016. DOI: 10.48550/ARXIV.1607.00653. URL: https://arxiv.org/abs/1607.00653.

[43] Elior Cohen. *Node2Vec - Github.* URL: https://github.com/eliorc/node2vec.

[44] Wikipedia. *Pearson correlation coefficient.* URL: https://en.wikipedia.org/wiki/Pearson_correlation_coefficient.

[45] Wikipedia. *Explainable AI.* URL: https://en.wikipedia.org/wiki/Explainable_artificial_intelligence.

[46] Github Distributed Machine Learning Community. *XGBoost.* URL: https://github.com/dmlc/xgboost.

[47] Thomas N. Kipf and Max Welling. “Semi-Supervised Classification with Graph Convolutional Networks”. In: *CoRR* abs/1609.02907 (2016). arXiv: 1609.02907. URL: http://arxiv.org/abs/1609.02907.

[48] William L. Hamilton, Rex Ying, and Jure Leskovec. “Inductive Representation Learning on Large Graphs”. In: *CoRR* abs/1706.02216 (2017). arXiv: 1706.02216. URL: http://arxiv.org/abs/1706.02216.

[49] *Deep Graph Library.* URL: https://www.dgl.ai/.