Feature Generation for SME Behavioral Credit Scoring Model Using Graph Embeddings

by

Kerem Kaşıkcı

A Dissertation Submitted to the
Graduate School of Sciences and Engineering
in Partial Fulfillment of the Requirements for
the Degree of

Master of Science

in

Computer Science and Engineering

KOÇ ÜNİVERSİTESİ

September 18, 2024

Feature Generation for SME Behavioral Credit Scoring Model Using Graph Embeddings

Koç University

Graduate School of Sciences and Engineering

This is to certify that I have examined this copy of a master's thesis by

Kerem Kaşıkcı

and have found that it is complete and satisfactory in all respects,
and that any and all revisions required by the final
examining committee have been made.

Committee Members:

_____

Prof. Mehmet Gönen (Advisor)

_____

Assist. Prof. Barış Akgün

_____

Prof. Mehmet Güray Güler

Date: _____

To my beloved family.

ABSTRACT

Feature Generation for SME Behavioral Credit Scoring Model Using Graph Embeddings

Kerem Kaşıkcı

Master of Science in Computer Science and Engineering

September 18, 2024

Banks lend credit regarding customers' credit scores from in-house credit risk models. Corporate customers hold greater risk and are evaluated with already high-performing models. In this thesis, corporate customers' financial interactions (money transfers) with each other are used to develop an additional feature set to improve the behavioral credit risk scoring model of small and medium-sized enterprises. Graph representation learning does a good job in terms of extracting the essence of such relationships and projecting them into a multidimensional space. For this purpose, a graph of enterprises is constructed where they are nodes and their sum of money transfer amount over time are edges. An inductive graph representation learning algorithm, GraphSAGE, is employed. Therefore, the graph is created with directed and weighted edges and reduced to a strongly connected graph. Then, the algorithm is run in various settings to achieve the best performance. A credit scoring pipeline using different machine learning algorithms is added to improve the current model with these new embedding features. QNB Finansbank's real banking data for 2022 is used to perform this study. After looking at the results of the computational experiments, the most important contribution came from the concatenation of multiple embeddings generated with different aggregator functions. Starting from here, a potential economic savings is calculated. Additionally, it is seen that the new architecture improves the credit scoring of nodes with either low transfer amount or number of edges the most.

ÖZETÇE

Çizge Gösterimlerini Kullanarak KOBİ Müşterilerin Kredi Modelleri için Öznitelik Yaratımı

Kerem Kaşıkcı

Bilgisayar Bilimleri ve Mühendisliği, Yüksek Lisans

18 Eylül 2024

Bankalar, müşterilerinin şirket içi kredi riski modellerinden gelen kredi puanlarına göre kredi verirler. Tüzel müşteriler daha fazla risk taşır ve zaten yüksek performans gösteren modellerle değerlendirilirler. Bu tezde, tüzel müşterilerin birbirleriyle olan finansal etkileşimleri (para transferleri), küçük ve orta ölçekli işletmelerin davranışsal kredi riski puanı modelini iyileştirmek için ek bir öznitelik seti geliştirmek için kullanılmıştır. Çizge temsili öğrenimi, bu tür ilişkilerin özünü çıkarma ve bunları çok boyutlu bir uzaya yansıtma açısından iyi bir iş çıkarır. Bu amaçla, düğümleri işletmeler ve zaman içindeki para transfer tutarlarının toplamı kenarları olan bir çizge oluşturulmuştur. Tümevarımsal bir çizge temsili öğrenme algoritması olan GraphSAGE kullanılmıştır. Bu nedenle, çizge yönlü ve ağırlıklı kenarlarla oluşturulur ve güçlü bir şekilde bağlı bir çizgeye indirgenmiştir. Daha sonra, algoritma en iyi performansı elde etmek için çeşitli ayarlarda çalıştırılmıştır. Mevcut modeli bu yeni gösterim öznitelikleriye iyileştirmek için farklı yapay öğrenme algoritmaları kullanan bir kredi puanlama akışı eklenmiştir. Bu çalışmayı gerçekleştirmek için QNB Finansbank'ın 2022'ye ait gerçek bankacılık verileri kullanılmıştır. Hesaplamalı deneylerin sonuçlarına bakıldığında, en fazla katkı, farklı toplayıcı işlevleriyle üretilen birden fazla gösterimin birleştirilmesinden gelmiştir. Buradan hareketle bir potansiyel ekonomik tasarruf hesaplanmıştır. Ayrıca yeni mimarinin, düşük transfer miktarına veya kenar sayısına sahip düğümlerin kredi puanlamasını en çok iyileştirdiği görülmektedir.

ACKNOWLEDGMENTS

The most special thanks to my advisor Prof. Mehmet Gönen for his support during every phase of my master's education.

A special thanks to Emre Atan and Cem Kıraç from QNB Finansbank R&D for their support with the data.

Thanks to QNB Finansbank R&D for letting me peform this study using their data.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABBREVIATIONS

ANN	Artificial Neural Networks
AUROC	Area Under the Receiver Operating Characteristic Curve
CNN	Convolutional Neural Network
DT	Decision Tree
GAE	Graph Autoencoders
GAT	Graph Attention Network
GCN	Graph Convolutional Network
GGNN	Gated Graph Neural Network
GNN	Graph Neural Network
GRL	Graph Representation Learning
GRU	Gated Recurrent Unit
k-NN	k-Nearest Neighbors
LASSO	Least Absolute Shrinkage and Selection Operator
LDA	Linear Discriminant Analysis
LR	Logistic Regression
MLP	Multilayer Perceptron
NLP	Natural Language Processing
NPL	Non-Performing Loan
NB	Naive Bayes
PCA	Principal Component Analysis
ReLU	Rectified Linear Unit
RF	Random Forest
ROC	Receiver Operating Characteristic Curve
SHAP	Shapley Additive Explanations
SVM	Support Vector Machine
XGB	Extreme Gradient Boosting

Chapter 1

INTRODUCTION

Loan allocation is one of the core challenges for the banking sector, and it ranks among the top issues. A successful loan allocation is considered to be one that is not defaulting in the case of customers being unable to fulfill payments. On the one hand, one such allocation brings a slight return as the interest rate. On the other hand, in case of a default, together with the return, a significant part of the loan is also lost, which is much larger as well. This effect gets even more exaggerated when the concerned party is the corporate customers because their numbers shrink regarding the individual customers, yet their balances become immense.

Especially after 2000, banks started to deploy in-house developed models using machine learning algorithms for credit scoring, or credit scorecards, for various reasons. The most critical reasons are the increased availability of modeling software and algorithms, compliance with the Basel Accords, efficiency, objectiveness, increased profitability, and improved customer experience properties of machine learning models (Basel Committee on Banking Supervision, 2017; Siddiqi, 2012). Another aspect is that the in-house data usually best reflects a bank's customer portfolio's risk profile. On top of this, these data are exploited in many ways, making the models perform already well. While exploring new ways of benefiting from customer data is challenging, it always brings enormous value (Siddiqi, 2017).

For the QNB Finansbank, corporate customers' financial interactions with each other remained unexplored so far in terms of being included in a behavioral credit risk scoring model. Therefore, in this thesis, graph representation learning is used since it is well known to represent entity relationships with each other.

To prepare a graph representation learning data set, first, the null features are im-

puted using machine learning algorithms. Then, a graph with companies as nodes and their money transfer amounts as edge weights and directions is constructed. Afterward, the GraphSAGE algorithm is used to embed the relationships as a new input space (Hamilton et al., 2017). As the authors state, it is an algorithm for "Inductive Representation Learning on Large Graphs" known for its ability to scale to large graphs while learning node representations efficiently. The inferred representations are used as a new input space to improve the credit scoring of small and medium-sized enterprises using machine learning.

1.1 Organization of the Thesis

Chapter 2 provides a literature review of the general structure of credit scoring and graph learning applications. Chapter 3 gives the theoretical background of classical machine learning algorithms used in the thesis. Chapter 4 provides a theoretical basis for the graphs and the GraphSAGE algorithm. Chapter 5 details the data set, performance metrics, the methodologies used in experiments and preprocessing, and experiment and performance analysis results. Chapter 6 concludes the thesis by summarizing all the inferences reached during the work and prospective future work directions.

Chapter 2

LITERATURE REVIEW

In this chapter, prevalent methodologies in credit scoring will be summarized first, and then recent studies in this area with graph neural networks and graph representation learning will be covered.

2.1 Credit Scoring

The credit scoring literature dates back to the 1950's. However, the essence of the modern works will be aimed to be conveyed. Dastile et al. (2020) reviews 74 primary journal and conference articles published between 2010 and 2018 in a well-designed structure. For this reason, this work constitutes the backbone of this section.

A scorecard aims to decide whether borrowers can be lent without any unfair prior judgments. They are mainly established with two sets of data: bank/finance institution and bureau information. The former reflects the data owned by the institution: demographics, financial product usage, behaviors, and any delinquency of the institution's loans. The latter reflects a summary of the customer within the whole financial system shared with the institutions through queries. Using this information, a model is constructed by looking, e.g., 24 months prior. For target definition, any more than 90 days overdue is considered a bad borrower and a good borrower, if not any. After model construction, a lending score cut-off is determined according to a statistical test such as the Kolmogorov-Smirnoff test. The traditionally used algorithm is the logistic regression (LR) due to its simplicity and transparency.

In the following subsections, frequently used methodologies in credit scoring will be mentioned, along with brief descriptions. These will cover feature selection, feature engineering, machine learning models, evaluation metrics, and observations

and limitations.

2.1.1 Feature Selection

Feature selection is the process that reduces the model input features into a subset of itself. Studies show that it does not always increase performance, but removing redundant features can. It is divided into three categories: filtering, wrapper, and embedded methods.

The filtering methods select features according to a univariate analysis without a predictor. One of the most popular is the F-score, which indicates the discrimination of a feature based on positive and negative samples. Another is rough set theory, where features are grouped according to their indiscernibility relations, and minimum redundancy might be achieved by selecting one feature from each group.

The wrapper category uses a prediction and selects features according to a performance indicator, e.g., the area under the receiver operating characteristic curve (AUROC). They perform better but demand more computation power since each subset is scored individually. One example is stepwise selection, which uses linear regression and p-value. It either adds features one by one until there is a limited performance increase, named forward selection. Also, it might work as subtracting features from the whole set until a performance degradation occurs, known as backward feature elimination. Another is the combination of both, which is stepwise feature selection. Another wrapper method is the genetic algorithm. It works with the same logic as in biology with its selection, mutation, and crossover abilities of chromosomes, which, in this case, are features.

For the embedded methods, the most well-known example is the least absolute shrinkage and selection operator (LASSO). It optimizes a learning objective together with an ℓ_1-regularization of its parameters.

2.1.2 Feature Engineering

Feature engineering creates a new set of features by putting the features into various algorithms to increase or decrease the set size or boost the underlying relationships.

The ultimate goal is to improve model performance. Popular feature engineering methods are principal component analysis (PCA), autoencoders, and linear decomposition analysis (LDA). PCA tries to compress the feature space into a much lower dimension. It uses eigenvectors that are orthogonal to each other from the decomposition of the feature covariance matrix. Autoencoders are a specific type of neural network that first lowers the dimensions of the input through encoding layers and then tries to reconstruct its input through decoding. They might be used for efficient dimensionality reduction, denoising the inputs. LDA is similar to PCA, but it learns the eigenvectors to maximize the between-class variance to the within-class variance. Namely, it also tries to learn class-discriminative features using class (target) information. Feature extraction techniques' usage frequencies, reported by Dastile et al. (2020), are given in Table 2.1.

Table 2.1: Usage frequency of feature extraction techniques in credit scoring.

Algorithm	Frequency (out of 74)
Rough Set	6
Genetic Algorithm	4
Stepwise	4
PCA	4
LDA	2
LASSO	1
Auto-encoder	1

2.1.3 Machine Learning Models

Credit scoring is a supervised learning problem, specifically, a binary classification that determines whether the borrower can pay the loan. Therefore, any binary classification model might be used that takes an input space $\boldsymbol{x} \in \mathbb{R}^D$ and maps it into a pre-defined target $y \in \{-1, +1\}$. Again, the frequently used ones will be listed, and decision tree (DT), random forest (RF), extreme gradient boosting

(XGB), and LR will be detailed in Chapter 3.

Statistical models are LDA, LR, and Naïve Bayes (NB). NB uses Bayes' Theorem while using prior probabilities as Pr($class$) and the likelihood of features given class Pr($feature|class$). The posterior likelihood of a class given features Pr($class|feature$) is calculated from there. Its naive feature comes from the assumption that features are mutually independent.

Machine learning models are k-nearest neighbors (k-NN), support vector machine (SVM), artificial neural networks (ANN), boosting, bagging, RF, and DT.

k-NN takes an entity's feature vector $\boldsymbol{x}$ as the input and determines its closest neighbors according to Euclidean or Mahalanobis distance. Then, it chooses the entity label as most of its neighbors' class labels. In linear SVM, a decision boundary is calculated, and then its margin is tried to be maximized. For non-linear SVM, a kernel function is applied first to move the feature space to a higher-level representation. ANNs are inspired by biological neurons aiming to capture non-linear relationships based on their activation with specific inputs. The weights in its hidden and output layers are learned through a backward differentiation called back-propagation.

The boosting methods start with a weak learner; the following model fittings focus on erroneous examples by assigning them higher weights. In the bagging techniques, weak learners are trained from different sample sets derived with bootstrapping, where each sample might be resampled again. In the below Table 2.2, the most frequently used supervised learning algorithms' frequencies reported by Dastile et al. (2020) are provided. Since more than one algorithm can be used in one project, these frequencies are not mutually exclusive.

2.1.4 Evaluation Metrics

Evaluation metrics describe a model's predictive power. True positives (TP) and true negatives (TN) are the numbers of correctly classified positive and negative samples, respectively. False positives (FP) and false negatives (FN) indicate the number of incorrectly classified samples. Accuracy, or percentage correctly classified,

Table 2.2: Usage frequency of supervised learning algorithms in credit scoring.

Algorithm	Frequency (out of 74)
Support Vector Machine	47
Logistic Regression	40
Artificial Neural Network	35
Decision Tree	25
Boosting	17
Random Forest	14
Bagging	13
k-Nearest Neighbors	12

is the percentage of truly predicted samples to the whole sample set. Sensitivity, also known as recall or true positive rate, emphasizes not missing the positive samples. Specificity is the opposite; it measures the success of capturing the negative samples. Precision measures how correctly a model predicts positive examples. F_1-Score or F-measure is the harmonic mean precision and recall and offers an overall performance measure for imbalanced data sets.

The receiver operating characteristic (ROC) curve shows the trade-off between true positive and false positive rates for each threshold value. The area under the ROC curve (AUROC) is used for the discriminative power of the model. It ranges between 0 and 1, with 0.5 equal to completely random predictions. Their formulations are given with the below equations, and usage frequencies reported by Dastile et al. (2020) are indicated in Table 2.3; these numbers are not mutually exclusive. Each signifies different characteristics and, therefore, should be chosen purposefully.

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{FP} + \text{TN} + \text{FN}} \tag{2.1}$$

$$\text{Type I Error} = \frac{\text{FP}}{\text{FP} + \text{TN}} \tag{2.2}$$

$$\text{Type II Error} = \frac{\text{FN}}{\text{TP} + \text{FN}} \tag{2.3}$$

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{2.4}$$

$$\text{Sensitivity/Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{2.5}$$

$$\text{Specificity} = \frac{\text{TN}}{\text{FP} + \text{TN}} \tag{2.6}$$

$$F_1\text{-Score} = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{2.7}$$

Table 2.3: Usage frequency of evaluation metrics in credit scoring.

Evaluation Metric	Frequency (out of 74)
Accuracy	47
AUROC	31
Type I	14
Type II	14
Sensitivity	13
Specificity	10
F_1-Score	4

2.1.5 Observations and Limitations

Some observations or limitations within the credit scoring literature are as follows:

- According to the Basel III Accord, banks must inform their customers why their applications have been rejected (Basel Committee on Banking Supervision, 2017). Only 8% of the studies considered a transparency technique.

- Such transparency obligation limits the use of deep learning algorithms because of their black-box nature. Therefore, they are used in at most 5 studies.

- Credit scoring data sets generally have a highly imbalanced target distribution because few people default, and most continue paying their debts. However, only 18% of the studies applied a balancing technique.

- Macroeconomic variables are not fed into the models because their necessity to be forward-looking casts an intrinsic hardship.

- Most often, the correlation between input features is not measured.

- Feature selection or feature space reduction is chosen over new feature engineering.

- Time and model complexity are not thought about.

- Outlier detection and variable distribution observation are not performed.

- Cut-off determination is often ignored.

2.2 Graph Learning in Banking

In this section, the literature on graph learning in banking is reviewed over existing studies similar to this thesis.

2.2.1 Deeptrax: Embedding graphs of financial transactions

Khazane et al. (2019) introduced a representation learning for bipartite graphs for merchants and accounts using credit card transactions. The learned representations are used in a downstream task, a fraud detection model. It is showed that the model can learn spending behavior, geography, and merchant industry/category patterns.

An implicit account/merchant bipartite graph is split into two graphs of merchants and accounts, which has reportedly improved computational time and flexibility. The merchant graph has weighted edges according to shared accounts that have transacted on both within a time frame. They have worked with two data sets, one brand-level with 10^4 nodes and 10^7 edges; the other was a raw merchant graph

with 10^6 merchants and 10^9 edges. The embeddings are trained using negative sampling, where node pairs are positive and randomly sampled nodes are negative. The idea that if the number of random walks per node is sufficient, the expected average walk length will converge to the shortest path between nodes is leveraged.

In link prediction, the architecture scored 0.90 AUROC and 0.67 F_1-score. A price point direction in visualizing the embeddings similar to word embeddings is explored, e.g., Banana Republic and Gap. When the raw merchant data set embeddings is included within a fraud detection model, it brought a 1.0% increase in the AUROC score. When a separate multilayer perceptron is trained for detecting fraudulent transactions, it achieved a 5.2% improvement in area under the precision-recall curve.

2.2.2 Risk assessment for networked-guarantee loans using high-order graph attention representation

By the definition of a guarantee loan, one company might be the guarantor of another to undertake the burden in case of an inability to repay. For the companies' side, it eases their reach to a loan. On the other hand, when looking at the complete picture of guarantor relations, companies may form dangerous formations that might evoke a chain of failing to repay, similar to an infection spread. These patterns are hard to detect using classical machine-learning techniques. To overcome such difficulties, Cheng et al. (2019) proposed a high-order graph attention representation (HGAR) method.

The network is defined as directed according to being a guarantor or a guarantee. In this architecture, latent embeddings of nodes are fed into an attention layer in neural networks that learns higher-level representations. For this purpose, first, the inputs are multiplied by a shared linear transformation matrix to project them into higher-order. Then, a shared attention mechanism is applied on top of it. The first-order and higher-order attentional features are concatenated for the credit default prediction process. The architecture is tested using data from an Asian commercial bank, with almost 113,000 nodes and 125,000 edges.

Compared with other widely used graph neural networks (GNN), this method yielded almost a 4% AUROC score increase. Another experiment shows that the higher-order transformation and the attention mechanism contribute to the performance. Further exploration revealed that the method can isolate previously defaulted firms, whereas other GNNs struggle to do so.

2.2.3 EWS-GCN: Edge weight-shared graph convolutional network for transactional banking data

Sukharev et al. (2020) tried to adapt modern deep learning approaches to the credit scoring procedure using client transactions. A new graph neural network is developed, edge weight-shared graph convolutional network (EWS-GCN), which is built upon the idea of combining graph convolution and recurrent neural networks with an attention mechanism. The network has the distinctive properties of a special attention mechanism and weight-sharing for convolutional layers, allowing the building of deeper architectures.

To provide more details, a large European bank's client data is used, which consists of purchases and transfers. The data constituted about 10^6 nodes and 10^{10} edges. Subgraphs with two-hop neighborhoods are used. The basic innovation in the model is to weigh each node feature vector channel separately, applying a linear transformation to both node and edge features. The attention mechanism is based on edge features, and the rectified linear unit (ReLU) activation function is introduced to nullify negative values, reducing computational complexity through sparsification. In addition, using a non-linear learnable aggregation of node and neighbor representations via a gated recurrent unit (GRU) cell, similar to gated graph sequence neural networks (GGNN) is proposed (Li et al., 2015). More than 7,000 features are generated using sum, average, median, minimum, maximum of transactions and their properties, later reducing them to 1,000 best features. In the graph, thresholding is applied on edges to remove weak connections and then treated the graph as unweighted. The different graph features minimum/maximum/average/etc. of PageRank, node degree, and betweenness centrality brought 500 new features.

When trained on credit scoring, their proposed model reached an AUROC 0.89% higher than the graph attention network (GAT), and edge-enhanced graph neural network (EGNN) benchmarks (Velickovic et al., 2017; Gong and Cheng, 2019).

2.2.4 Linking bank clients using graph neural networks powered by rich transactional data

Shumovskaia et al. (2021) focused on predicting if any pair of customers will interact with each other by formulating the problem as a link prediction. In this regard, a graph neural network is proposed that can use the topological information and the time series from the nodes and edges of a large European bank data set. Then this infrastructure is used to improve the quality of a credit scoring model.

To give more detail, the SEAL framework from Zhang and Chen (2018) is used as a base graph neural network. The SEAL is designed to learn general graph structure features from local enclosing subgraphs. Shumovskaia et al. (2021) adapted it with a recurrent neural network (RNN) to work with time-series data. The RNN's output for link probabilities is used as an attention mechanism. Also, its hidden layer outputs, "embedded transactions," are used as node features in the graph neural network. The RNN is pre-trained with the credit scoring target. The sortpooling process of SEAL is altered into just taking the embeddings of the related node pairs that are input to link prediction, to reduce the number of parameters and computational time. For the same reason, 2-hop neighborhood subgraphs are used for all the trials. Additional trials are made with the pooling methods and node features, which will not be detailed here.

In the credit scoring case, a simple RNN is deployed, a GRU cell followed by linear layers as a baseline, scoring 85.5% AUROC. Adding GCN increased the model's performance by 0.4% AUROC score. However, GCN has a problem with treating every neighbor equally, lacking an attention mechanism. Therefore, when the proposed infrastructure 2-SEAL-RNN is used, a 0.7% AUROC score increase is provided.

2.2.5 Graph neural network with self-attention and multi-task learning for credit default risk prediction

Li et al. (2022) proposed a novel GNN model with self-attention and multi-task learning (SaM-GNN). The primary focus of the architecture is to better fill the missing entries with the aggregations from close neighbors.

In this framework, first, the attributes pass through numerical and categorical self-attention layers in order to generate feature representations. Concatenated representations are conveyed to the graph convolution layer. Here, subgraphs are defined according to whether customers share a common value within a certain set of features. Customers' number of common features constitute the edge weights as well. Also, this feature set is defined according to distinct values and missing ratio thresholds. Graph convolution aggregates the neighbors' information and updates an intermediate (representation) vector. A decoder module of two fully connected layers takes these inputs to capture a better representation. In order not to lose any information, all the raw numeric vectors and the intermediate vector are concatenated. Then, they are sent to a classification layer for loan allocation. Again, this layer consists of fully connected layers. In addition, a joint learning loss and a mean square for vector reconstruction and cross-entropy for learning tasks are used. The technique is applied to two public data sets with high-dimensionality and high-volume features.

When it is benchmarked against LR, DT, XGB, convolutional neural network (CNN), and multilayer perceptron (MLP), it has brought 26% additional AUROC in one data set and 1% in the other. Upsampling the positive examples improved the results between 1-3% AUROC.

2.2.6 CDGAT: A graph attention network method for credit card defaulters prediction

Wu et al. (2023) have proposed a new architecture, a graph attention network for credit card defaulters (CDGAT), which is based on transaction and neighborhood embeddings. The architecture first uses an amount-bias sampling (AbS) strategy

to derive a subgraph for each client. Second, it aggregates the neighbors' features regarding their attention weights. The project has been implemented on a data set from the Industrial and Commercial Bank of China (Macau) Limited (ICBC), seemingly it outperforms all the benchmarks consisting of classical machine learning, deep learning models, and GNNs.

To give more detail, the architecture first applies a GRU with attention to create customer transaction embeddings from raw transactions. Then, the transaction network of transfers is sampled into subgraphs with AbS, which weighs the transactions regarding that amount while sampling for each customer. This reduces both memory and computational time requirements. Afterward, the neighbors' features are aggregated according to a graph attention model, an extension of the GAT method to use multi-head attention to aggregate the features directly from the neighbors in different neighborhoods. After concatenating the transactional and neighborhood embeddings, they are fed into a fully connected neural network in order to model credit card defaults. To mention the size of the data set, there are almost 100,000 customers, 15,000,000 transactions, and 3,000,000 transfer records. The GRU with attention mechanism is chosen according to the credit default model performance among many competitors ranging from the fully connected neural network, CNN, RNN, long short-term memory, and self-attention network (LeCun et al., 2015; Vaswani et al., 2017).

The findings show that increasing the neighborhood level from 1 to 3, with or without AbS, increases the model's learnability. Also, 80% for 1-hop, 60% for 2-hop, and 30% for 3-hop neighborhood sampling is enough for the model to reach its best performance. When the architecture is compared with the classical machine learning and deep learning methods; LR, SVM, light gradient boosting machine (LGBM) of Ke et al. (2017), and embedding-transactional-RNN of Babaev et al. (2019), to name some, it surpasses the second-best competitor by almost a 4.5% AUROC score, reaching 83.4%. When compared with other GNNs; GCN, GraphSAGE, GAT, simple graph convolution (SGC), and graph isomorphism network (GIN), the difference is 1.1% AUROC with second best (Wu et al., 2019; Xu et al., 2018). In addition, four of them threw memory overflow errors with 128 GB RAM, except for

GraphSAGE and CDGAT.

2.2.7 On the combination of graph data for assessing thin-file borrowers' creditworthiness

Thin-file borrowers are customers who do not have much credit history. Therefore, their credit score is hard to measure. For this reason, they are mostly left out of the loan system. In short, Muñoz-Cancino et al. (2023) used feature engineering, graph embeddings, and graph neural networks from various data sources on the entire society of a Latin American country. The framework is simulated for both application and behavior scoring and for both individuals and businesses. Improvements are achieved especially in the business application scoring, which aligns with the purpose of the project.

To give more details, a collection of various data sources is used, ranging from marriages, transactional services, enterprise ownership, parents and children, and employment networks. A data set of around 7,600,000 people and 245,000 businesses is created as a combination of these data sets. In addition to these raw data sets, some additional data sets are generated: NodeStats includes node centrality statistics, egoNet aggregations are the weighted average and standard deviations of the neighbors' NodeStats, graph embeddings of family and enterprise owner-worker network using Node2Vec from Grover and Leskovec (2016), GNN (either variational graph autoencoders (GAE) or GCN) features of the same networks where the target being defaulter, non-defaulter, or unbanked (Kipf and Welling, 2016b). Then a variable selection is applied using feature importance (53% AUROC) and correlation (70%) thresholding for both intra-data sets and inter-data sets. The remaining variables are fed into a gradient-boosting algorithm to learn credit scoring.

Results of trials with different data set combinations show that business application scoring models AUROC increase by 9.0%, behavioral by 2.80%, individual application by 3.58%, and behavioral by 2.18%. Shapley Additive Explanations (SHAP) values help determining feature importance and performing sensitivity analysis (Lundberg and Lee, 2017). According to the SHAP values from the final models,

enterprise owner-worker network variables are the most important features.

2.2.8 Improved credit risk prediction based on an integrated graph representation learning approach with graph transformation

Shi et al. (2024) approached the problem of not having explicit customer interaction information differently. Shortly, the method finds a customer's closest neighbors using k-NN according to the distances of their attributes in order to construct implicit edges. Then, such graph formations are used to predict credit defaulters as supervised node classification. Four widely used open-access credit scoring data sets are used in order to measure their performance.

To provide more details, they tested the framework for various k-neighborhood values for k-NN. The results suggest that the best k values vary for each data set, therefore it has to be fine-tuned initially. One more dimension in this experimentation is the type of the GNN network; GCN, GraphSAGE, and GAT. Then, the champion k-NN-GNN setting is compared to benchmark models for each data set.

Looking at the AUROC scores, GCN surpasses the other GNNs, and k-NN-GCN tops all other benchmarks with 1-5% AUROCs. Another experiment has been ensembling multiple k-NN-GCNs with different k parameters. It brought a slight increase in performance, mostly below 1% AUROC. The last experimentation was the introduction of edge weights into GCN and GAT algorithms as inverse node distances. The results vary for different k values, but on average, there is a slight improvement.

Chapter 3

MACHINE LEARNING ALGORITHMS

This chapter examines the classical machine learning algorithms used in this thesis: decision tree, random forest, extreme gradient boosting, and logistic regression. In addition, evaluation metrics used for machine learning tasks are provided at the end of the chapter. Graph representation learning algorithms are discussed in the following chapter.

3.1 Decision Tree

First proposed by Breiman (2017) in 1984 originally, decision trees are most widely used as benchmarks due to their simplicity and explainability. It can work with both classification and regression tasks. A tree is a framework that starts from the whole data set and splits it into descendant subsets. This starting node with the entire data set is named the root node, which leads to the following nodes by binary splits. Eventually, they lead to the leaves, the ultimate decision-maker nodes in each tree branch. Each leaf is assigned a class label according to most classes in that terminal subset. The whole architecture looks like a tree growing downwards when it is visualized.

The algorithm revolves around three problems: how to choose a split, when to stop splitting, and assigning each leaf to a class. Determining a split position is done by examining every variable and every split position within each node. A split is halfway through for each distinct value for continuous variables and each distinct value for categorical variables. Selecting the split according to Breiman (2017) is choosing the one that brings the most impurity decrease. A node is purest when it consists of only one class, and it gains impurity as it is contaminated more by other classes. To stop the splitting, a minimum impurity decrease threshold is

set. Other performance metrics might also be used to select splits, such as cross-entropy decrease and log-loss decrease. The impurity definition is given as the below equation, where $i(t)$ is the impurity of node t, where j is a class among all sets of classes $\{1, 2, \ldots, K\}$, and $\Pr(j|t)$ is the node proportion of class j.

$$i(t) = -\sum_{j=1}^{K} \Pr(j|t) \log \Pr(j|t) \tag{3.1}$$

In addition to the impurity decrease threshold, several other essential hyper-parameters control the growth of a tree according to preference. Maximum depth controls for how many vertical layers a tree can grow. A minimum number of samples before a split might be set to prevent it from splitting after reaching a certain number of samples. Similarly, a minimum number of samples for a leaf after splitting can be appointed. Again, a certain number of features might be set for the split search.

The regression task is similar yet simpler. There are no classes; therefore, one cannot assign class weights. The typical loss function is the residual sum of squares, as shown in Equation 3.2. In this equation, y_i is an actual target value, and $f(\boldsymbol{x}_i)$ is the model output for input $\boldsymbol{x}_i$ for $i \in \{1, 2, \ldots, N\}$.

$$L = \sum_{i=1}^{N} (y_i - f(\boldsymbol{x}_i))^2 \tag{3.2}$$

3.2 Random Forest

Random Forest (RF) is an ensemble learning adaptation to the DTs. Ensemble learning uses numerous different models to use each one's strength and, therefore, achieve better generalization. Ho (1995) made their first introduction to overcome the DTs' inability to grow complex trees without overfitting. Overfitting is the case of a model learning the patterns in its training data very well so that its performance on new data degrades. Breiman (2001) has adapted bagging and random selection of the features to RFs, and it is widely used. Therefore, this version will be mentioned.

An RF is a collection of decision-tree classifiers $h_\ell(x)$; where $\ell \in \{1, 2, \ldots, L\}$, L is the number of decision-tree classifiers. $\boldsymbol{x}$ is the input, and y is the output. The

margin function is defined as:

$$mg(\boldsymbol{x}, y) = av_\ell I\left(h_\ell(\boldsymbol{x}) = y\right) - \max_{j \neq y} av_\ell I\left(h_\ell(\boldsymbol{x}) = j\right) \tag{3.3}$$

where $I(\cdot)$ is the indicator function, which returns 1 if the statement inside is true and 0 otherwise. av_ℓ is the average over all classifiers ℓ. The margin is how larger the correct predictions are from the wrong ones for the given $\boldsymbol{x}, y$. From there, the generalization error is the probability that the margin is less than zero for the given $\boldsymbol{x}, y$ space, where $h_\ell(X) = h\left(\boldsymbol{x}, \Theta_\ell\right)$.

$$PE^* = P_{x,y}(mg(\boldsymbol{x}, y) < 0) \tag{3.4}$$

RFs follow the Strong Law of Large Numbers according to Breiman (2001)'s theory; as with the sufficient numbers of trees in a forest, for almost all sequences of Θ, PE^* converges to below equation where Θ_ℓ is the random vector generated for learner h_ℓ and $h_\ell(x) = h\left(x, \Theta_\ell\right)$. This is a limitation on its generalization error, leading to better generalization.

$$P_{x,y}\left(P_\Theta(h(\boldsymbol{x}, \Theta) = y) - \max_{j \neq y} P_\Theta(h(\boldsymbol{x}, \Theta) = j) < 0\right) \tag{3.5}$$

Random selection of features limits the correlation of the learners while preserving their strength. The bagging or bootstrapping aggregation generates different random training sample sets with replacement. It is claimed to increase the model accuracy and can be used to give ongoing estimates of the generalization error, correlation, and strength.

On top of the hyperparameters described for the decision tree, the number of decision trees to grow and whether to bootstrap are the random forest hyperparameters.

3.3 Extreme Gradient Boosting

Extreme Gradient Boosting, or XGBoost (XGB), is another ensemble learning adaptation (Chen and Guestrin, 2016). Boosting and gradient descent are the core differences that give its name. Unlike bagging, boosting trees do not grow independently

but sequentially based on their predecessors' errors so that they can reach the aim of minimizing the total error. Gradient descent is an optimization algorithm where steps are taken in the descending direction of the loss function, looking at its derivative (gradient) to minimize the error.

To mention the improvements to the gradient boosting methods in XGB, a regularization term is added to the learning objective to prevent unnecessary complexity and overfitting. Here, the regression trees are different from the decision tree regressors. They do not provide point estimates, but they provide continuous function outputs for each leaf. An entry's score on each leaf is summed to reach its target value. A greedy algorithm for tree-forming is introduced that speeds up the tree-structure evaluation. A factor scales each newly added tree's weight to prevent overfitting and make room for future trees, and this process is called shrinkage. RF's column subsampling is used in this algorithm as well.

An approximate algorithm is suggested for overcoming the default exact greedy algorithm's computation complexity to find the best splits. It proposes quantiles as candidate points and finds the best split looking at aggregate statistics around these points. However, suggesting candidate splits with unbalanced data sets becomes a problem. A weighted quantile sketch that supports merge and prune operations is deployed to handle this problem. Sparse columns also constitute a hardship for the same task. The prevalence of missing values or zeros might cause sparsity. Here, trees learn a default direction from the data, when the values are sparse, an entry follows these directions. Therefore, split finding gains sparsity awareness as well.

Data sorting is the most time-consuming task in tree-forming. To ease solving this problem, data is sorted and packed as blocks named compressed columns, which require a single run over the whole data. The blocks are stored in memory and called when necessary, which makes statistics collection parallelizable. Since fetching gradient statistics by row indices leads to non-continuous memory access, gradients are fed into separate thread buffers. This is called cache awareness and is especially beneficial when working with large data sets. Compressed data blocks are saved to the disk memory to process too large data sets and decompressed on the fly. This process is called out-of-core computation.

To mention the hyperparameters of the XGB, the learning rate is an important addition to the preceding algorithms. To put it in simple terms, it scales the contribution of each boosting tree to a prediction

$$\hat{y}^{(t+1)} = \hat{y}^{(t)} + \eta h^{(t)} \tag{3.6}$$

where $\hat{y}^{(t)}$ is the output prediction, $h^{(t)}$ is the predictor learned at boosting round t, η is the learning rate.

Additionally, thresholds for minimum loss reduction or minimum sum of second-order gradients can be set while creating child nodes to prevent overfitting. Also, ℓ_1 and ℓ_2 regularization terms can be added to weights for the same reason. These regularizations penalize either the sum of absolute values or the sum of square values of the parameters, respectively.

$$\ell_1 \text{ Regularization Term} = \lambda \sum_j |\beta_j| \tag{3.7}$$

$$\ell_2 \text{ Regularization Term} = \lambda/2 \sum_j \beta_j^2 \tag{3.8}$$

In this generic equation, λ is the regularization coefficient, and β_j are the model parameters.

3.4 Logistic Regression

Logistic regression is a statistical model that uses linear combinations of the independent variables to output a log-odds or the logit function output. It is the inverse of the standard logistic function in Equation 3.9.

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{3.9}$$

The logit is defined by Barnard (1949):

$$\text{logit } p = \sigma^{-1}(p) = \ln \frac{p}{1-p} \text{ for } p \in (0, 1). \tag{3.10}$$

Therefore, the logistic function converts the logits into probabilities. This conversion makes it possible to use in binary classification.

$$p(x) = \frac{1}{1 + e^{-(x-\mu)/s}} \tag{3.11}$$

μ is the location parameter where $p(\mu) = 1/2$, and s is the scale parameter for x. With rewriting the terms as $\beta_0 = -\mu/s$ and $\beta_1 = 1/s$ the equation can be reformulated as

$$p(x) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x)}}. \tag{3.12}$$

With one more conversion and introduction of multiple independent variables x, we have:

$$\ln \frac{p}{1-p} = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + ... + \beta_D x_D \tag{3.13}$$

Now, the right-hand side looks like a linear regression, and the next step should be learning the best values of $\{\beta_0, \beta_1, \beta_2, \ldots, \beta_D\}$. The goodness of fit for the parameters is usually supervised by maximum likelihood estimation, which starts with logistic loss. A loss function describes the total diversion of predicted values from the actual ones. Let $p_i = \Pr(y_i = 1)$, and $1 - p_i = \Pr(y_i = 0)$ for a given independent values vector $\boldsymbol{x}_i$ and a dependent value y_i. Then the log loss for this point ℓ_i is equal to:

$$\ell_i = \begin{cases} -\ln p_i & \text{if } y_i = 1 \\ -\ln (1 - p_i) & \text{if } y_i = 0. \end{cases} \tag{3.14}$$

Since $0 < p_i < 1$, the loss ℓ_i is never zero. Therefore, it can be written as:

$$\ell_i = -y_i \ln p_i - (1 - y_i) \ln (1 - p_i) \tag{3.15}$$

The sum of this loss for all data points is known as the negative log-likelihood. It should be minimized, or its inverse positive log-likelihood can be maximized, known as maximum likelihood estimation. Positive log-likelihood is defined as

$$L = \sum_{i:y_i=1} \ln p_i + \sum_{i:y_i=0} \ln (1 - p_i) = \sum_{i=1}^{N} (y_i \ln p_i + (1 - y_i) \ln (1 - p_i)) \tag{3.16}$$

$$L = \ln \prod_{i=1}^{N} p_i^{y_i} (1 - p_i)^{(1-y_i)}. \tag{3.17}$$

Similar to XGB, a regularization type, coefficient, and a minimum threshold for loss reduction during training might be set. Additionally, a maximum number of

iterations for training might be predefined. Whether to equalize class weights with the inverse number of class samples and to add intercept parameter β_0 are some other hyperparameters.

Another important hyperparameter is the optimization-solving algorithm. Limited-memory Broyden–Fletcher–Goldfarb–Shanno with box constraints (L-BFGS-B) is a non-linear optimization solver intended to be used on large-scale cases where it is hard to construct the Hessian matrix (Zhu et al., 1997). Liblinear is designed to solve large-scale linear classification optimization problems like that of logistic regression and linear SVM (Fan et al., 2008). The stochastic average gradient (SAG) is a modification of the stochastic gradient. It keeps a memory of previous gradients and updates the parameters according to their average, therefore achieving faster convergence (Schmidt et al., 2017). Partly inspired by the SAG, the stochastic average gradient accelerated (SAGA) algorithm makes updates using a combination of both current and previous gradient differences and an average of all gradients.

3.5 Artificial Neural Networks

Although artificial neural networks (ANN) are not directly used in the machine learning tasks in this thesis, they constitute the fundamentals of the GNNs. ANNs are the popular machine learning architectures designed to tackle complex tasks inspired by biological creatures' learning procedures (Aggarwal, 2018). Biologically these procedures occur in neurons, cells in nervous systems that transmit information through each other. These cells build connections to one another in response to the strength of an external stimulant. This process is regarded as the root of the learning procedure in any living being.

Similar to a biological nervous system, an ANN consists of neurons. These neurons are the smallest computation units in an ANN. Usually, an ANN is made up of several layers of these neurons. The neurons take mathematical inputs $\boldsymbol{x}$ (similar to an external stimulant) and generate a response y to be passed to the next layer of neurons proportional to their connection weights $\boldsymbol{w}$, generally used as $y = \boldsymbol{w}^T\boldsymbol{x}$. These weights represent the connection strengths of the neurons, and

they are learned collectively through the training process with a technique called backward propagation of errors or "backpropagation".

An early, simple, and widely used form of an ANN is the multilayer perceptron (MLP) (Alpaydın, 2014). It is a nonparametric estimator that can be used both for classification and regression tasks. It consists of multiple layers of neurons: an input, an output, and one or more hidden layers. If we assume a single-layer perceptron for simplicity, it takes an input $\boldsymbol{x}$, where $x_j \in \mathbb{R}$, $j = 1, \ldots, D$ and D is the input dimension; returns a weighted sum of inputs y using weights $\boldsymbol{w}$, where $w_j \in \mathbb{R}$ and adding a bias term w_0 (see Figure 3.1a).

$$y = \sum_{j=1}^{D} w_j x_j + w_0 \tag{3.18}$$

Such a definition of a perceptron constitutes a hyperplane that can divide the input space into positive and negative half-spaces. Using a linear discriminant, the perceptron can distinguish two classes. If a threshold function $s(\cdot)$ is defined as

$$s(a) = \begin{cases} 1 & \text{if } a > 0, \\ 0 & \text{otherwise,} \end{cases} \tag{3.19}$$

then it can be deducted

$$\text{choose} \begin{cases} C_1 & \text{if } s(\boldsymbol{w}^T \boldsymbol{x}) > 0, \\ C_2 & \text{otherwise.} \end{cases} \tag{3.20}$$

Also, if the posterior probabilities are needed, the output can be converted as

$$o = \boldsymbol{w}^T \boldsymbol{x} \tag{3.21}$$

$$y = \text{sigmoid}(o) = \frac{1}{1 + \exp(-\boldsymbol{w}^T \boldsymbol{x})}. \tag{3.22}$$

A single-layer perception can only do linear approximation and does not imply any nonlinearity. Introducing the hidden layers into a single-layer perceptron and turning it into an MLP brings the power of nonlinear approximation, which can be seen in Figure 3.1b. Such a capability is empowered with the addition of more hidden layers. However, as the number of hidden layers grows, various problems

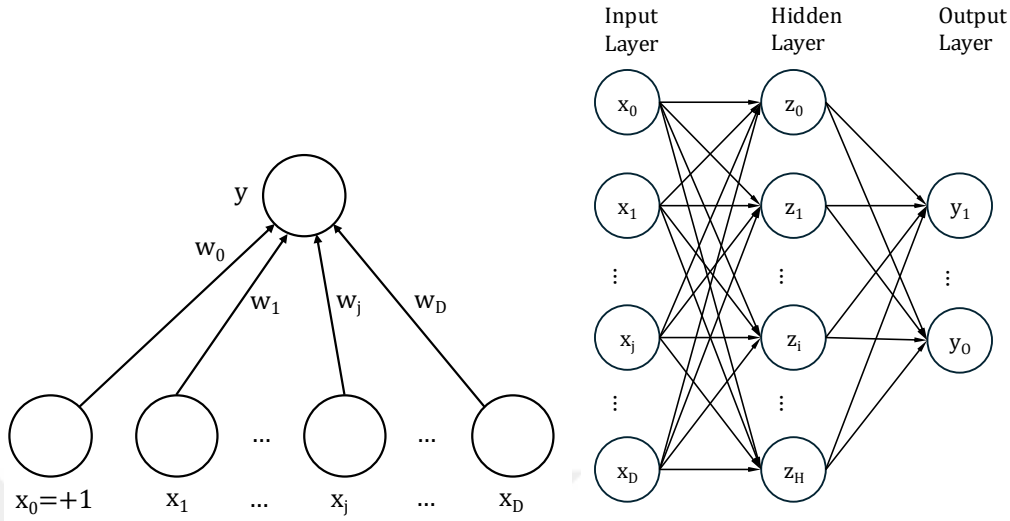


Figure 3.1: Perceptrons: (a) a simple perceptron with inputs x_j, bias unit x_0, weights w_j and output y, (b) a multilayer perception with the same inputs, an added hidden layer with hidden units z_j, and outputs y_o. D is the input, H is the hidden, and O is the output layer dimensions respectively.

emerge, such as vanishing/exploding gradients in backpropagation, overfitting, and computational complexity. Therefore, some countermeasures have been developed to mitigate these problems.

There are other specializations of ANNs as well. Convolutional neural networks are used to perceive image patterns using convolutional filters cascaded for different levels of perception. Recurrent neural networks are designed to handle tasks with sequential data, such as natural language processing (NLP) or time series. Autoencoders are used for unsupervised representation learning and are mostly used for dimensionality reduction and anomaly detection. Transformers are capable of handling long-range dependencies in data through their self-attention. They emerged and became known in NLP, affecting other areas such as computer vision and speech processing.

There are several essential parts of the ANNs that are worth further detailing, including activation functions, backpropagation algorithm, loss and optimization

functions, initializations, and learning rate decays.

3.5.1 Activation Functions

Activation functions play a significant role by introducing the non-linearity to ANNs. They possess different core properties of being asymptotical, computationally efficient, mimicking linear functions, etc. In this section, the neural network activation functions used in the architecture, rectified linear unit (ReLU), leaky ReLU, hyperbolic tangent (tanh), and the sigmoid will be introduced.

Sigmoid

The sigmoid (standard logistic or logistic sigmoid) function was introduced previously in Section 3.4.

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{3.23}$$

The same function is used in neural networks as activation functions before the emergence of the rectified linear units (Goodfellow et al., 2016).

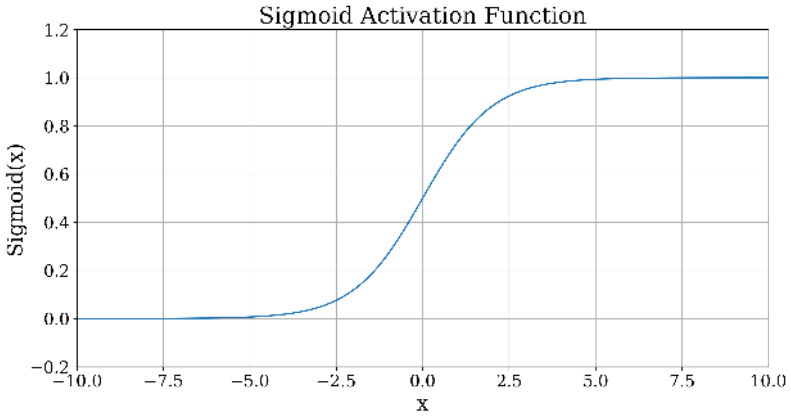


Figure 3.2: The sigmoid activation function

As can be seen from Figure 3.2, the function saturates for most of its domain. This emerges as a challenge for gradient-based learning, which may lead to slow convergence to the optimum loss value and vanishing gradients in the network as a

layer gets further away from the output layer. In addition, the exponential function is computationally expensive. For these reasons, their use as activation functions in feedforward networks is not preferred. However, they constitute a useful output unit with a suitable loss function compatible with gradient-based learning (Nwankpa et al., 2018).

Hyperbolic Tangent

Another sigmoidal function is the hyperbolic tangent or tanh:

$$\tanh(x) = \left(\frac{e^x - e^{-x}}{e^x + e^{-x}}\right). \tag{3.24}$$

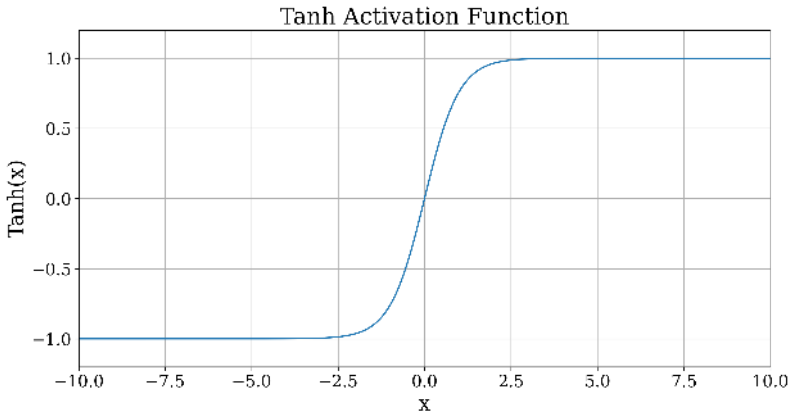


Figure 3.3: The tanh activation function

Tanh is closely related to the sigmoid because $\tanh(z) = 2\sigma(2z) - 1$. Since the tanh is more similar to an identity function around zero in the sense that $\tanh(0) = 0$ and $\sigma(0) = 1/2$, it is easier to train a network with a tanh activation function. However, it still cannot solve the problems of vanishing gradients and expensive computations. Despite their disadvantages as activation functions RNNs, many probabilistic models and some autoencoders use the sigmoidal functions (Nwankpa et al., 2018).

Rectified Linear Unit

Having mentioned the ease of training with an identity function, a rectified linear unit function resembles an identity even more and, therefore, is easier to train with less computational complexity.

$$\text{ReLU}(x) = \max\{0, x\} \tag{3.25}$$

Despite its similarity to a linear function, it still brings non-linearity due to not being activated with a negative input.

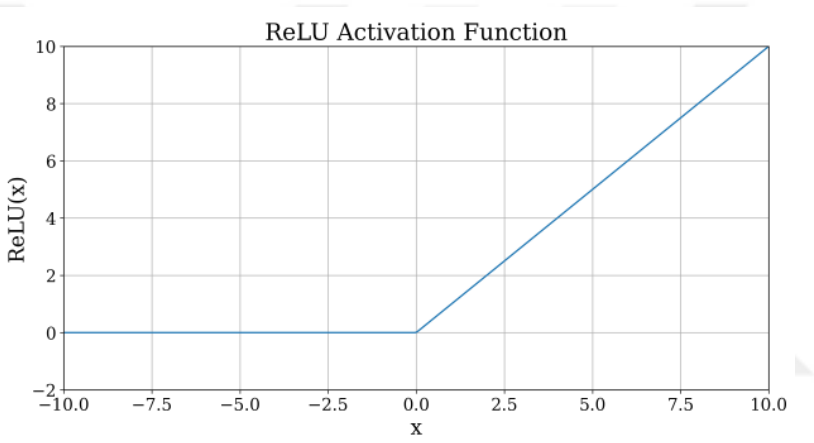


Figure 3.4: The ReLU activation function

One disadvantage of the ReLU is that it might overfit easier than sigmoidal functions. For this reason, they are used together with dropout (Srivastava et al., 2014). Another important disadvantage is that the input weights of a ReLU unit might be updated in an undesired way so that this unit outputs only zero values for any input point. Therefore, the learning from this unit is halted, and this phenomenon is also known as the "dying ReLU" problem. The leaky ReLU function has emerged to mitigate this issue (Nwankpa et al., 2018).

Leaky ReLU

Leaky ReLU is a modification to a ReLU where a nonzero slope α for $x < 0$. Therefore, it has nonzero gradients for the negative area to overcome the "dying

ReLU" problem (Nwankpa et al., 2018).

$$f(x) = \begin{cases} \alpha x & \text{if } x \leq 0 \\ x & \text{if } x > 0 \end{cases} \tag{3.26}$$

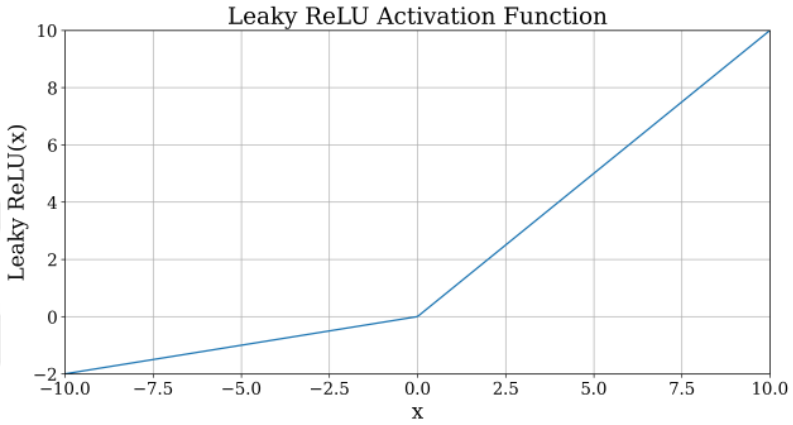


Figure 3.5: The Leaky ReLU activation function with $\alpha = 0.2$.

3.5.2 Backpropagation Algorithm

A multilayer perceptron's behavior is constituted by the learnable weights $\boldsymbol{w}$. Therefore, a neural network algorithm aims to optimize these weights through training to minimize prediction errors (Kubat, 2017). The backpropagation algorithm optimizes network weights by looking at how slightly changing weights affect the model prediction error using the chain rule from calculus. It has the same computational cost as calculating an output from an input vector, a forward pass. The error is computed at the output layer and flows backward as error contribution calculation and weight updates, "backpropagates".

If it is assumed that w_{ij} is the weight from neural network node i to j, a neuron j's output $h_j = f(a_j)$, E is an error function, η is a learning rate, then

$$\Delta w_{ij} = -\eta \frac{\partial E}{\partial w_{ij}} = -\eta h_i \delta_j \text{ where } \delta_j = e_j f'(a_j) = \left(\sum_k \delta_k w_{jk} \right) f'(a_j). \tag{3.27}$$

The chain rule is used upon the δ terms, also called "error signals" (Lillicrap et al., 2020).

3.5.3 Initializations

Weight initialization is appointing initial values to network weights before starting training. The choice of these initial values may affect the efficiency and effectiveness of the training procedure. The most basic form of initialization is to assign random values from a uniform or normal distribution.

Glorot (or Xavier) sampling is designed to prevent variance fluctuations in the back-propagation that may cause exploding or vanishing gradients within a neural network. The uniform version of the Glorot initialization is given in Equation 3.28. In this equation, $\mathbf{W}$ is the weights matrix, $U(-a, a)$ is a uniform distribution between $-a$ and a, n_j is the input size of layer j, n_{j+1} is the output size of layer j or input size of layer $j+1$.

$$\mathbf{W} \sim U\left(-\sqrt{\frac{6}{n_j + n_{j+1}}}, \sqrt{\frac{6}{n_j + n_{j+1}}}\right) \tag{3.28}$$

In the normal version of the Glorot initialization, the weights are sampled from a normal distribution

$$\mathbf{W} \sim N\left(0, \sqrt{\frac{2}{n_j + n_{j+1}}}\right) \tag{3.29}$$

where $N(\mu, \sigma)$ is a normal distribution with mean μ and standard deviation σ (Glorot and Bengio, 2010). A variance of $2/(n_j + n_{j+1})$ is aimed for both versions.

On the other hand, He initialization is designed for deep architectures with ReLU (or similar) activation functions (He et al., 2015). The main idea is similar: maintaining the network's variance, which might diminish due to ReLU not passing negative inputs. Only concerned with layer j's input size n_j, it aims for a variance of $2/n_j$. Its uniform and normal distribution variants are provided as follows:

$$\mathbf{W} \sim U\left(-\sqrt{\frac{6}{n_j}}, \sqrt{\frac{6}{n_j}}\right) \tag{3.30}$$

$$\mathbf{W} \sim N\left(0, \sqrt{\frac{2}{n_j}}\right) \tag{3.31}$$

3.5.4 Learning Rate Decay

According to Bengio (2012), the learning rate is "the single most important hyperparameter" in neural network training. For several reasons, the learning rate decay is the *de facto* technique during training. Generally, a large learning rate is initially set to accelerate learning and avoid any local minimum. It is also found that such a setting prevents memorizing noise in the data. Then, the learning rate is decayed so that the network hopefully converges to the global minimum and oscillation around the minimum is avoided. In addition, decay helps the network learn complex patterns (You et al., 2019).

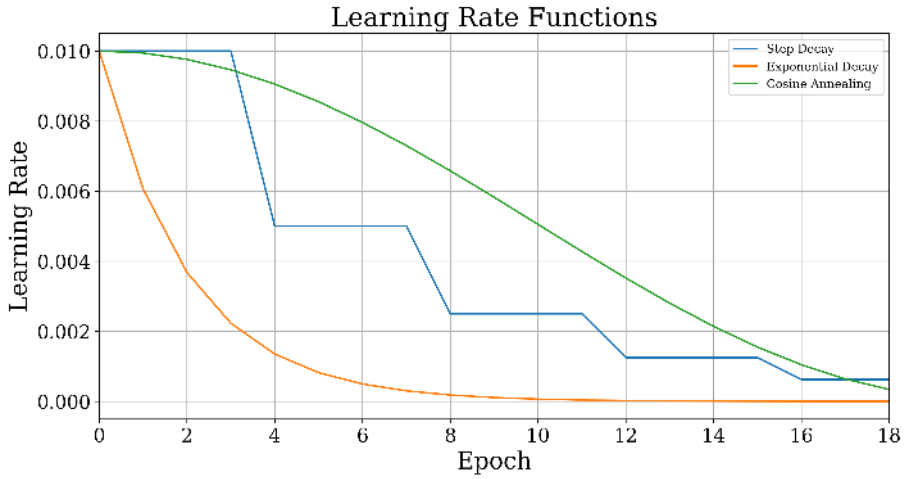


Figure 3.6: A step decay with drop factor = 0.5, step size = 4; an exponential decay with $\lambda = 0.5$; a cosine annealing with $\eta_{\min} = 0.0001$ over 18 epochs. η_0 or $\eta_{\max}$ is equal to 0.01 for all.

Some well-known learning rate decay functions are step, exponential decay, and cosine annealing.

$$\begin{aligned} \text{Cosine Annealing:} \quad & \eta_t = \eta_{\min} + \frac{1}{2}(\eta_{\max} - \eta_{\min})\left(1 + \cos\left(\frac{T_{\text{cur}}}{T_{\max}}\pi\right)\right) \\ \text{Exponential Decay:} \quad & \eta_t = \eta_0 \exp(-\lambda t) \\ \text{Step Decay:} \quad & \eta_t = \eta_0 \,(\text{drop factor})^{\left\lfloor \frac{t}{\text{step size}} \right\rfloor} \end{aligned} \tag{3.32}$$

In these equations, η_t is the learning rate at time step t, $\eta_{\min}$ is the minimum and

$\eta_{\max}$ is the maximum learning rate, T_{cur} is the current, and $T_{\max}$ is the maximum number of time steps. The λ is the exponential decay constant. The drop factor and step size are constants. The first one is similar to λ, indicates how much to modify the initial learning rate. The second indicates each time interval to modify it through flooring the dividing the t by it.

3.6 Evaluation Metrics

Evaluation metrics are an essential part of machine learning tasks in the sense of observing a model's current performance quantitatively. A brief overview of evaluation metrics in credit scoring is made in Section 2.1.4. Here, two different evaluation metrics used are covered in more detail, one for the models in the imputation phase and the other for credit scoring. Imputation fills the null values in the input features that might otherwise degrade a model's learning. In this thesis, the imputed variables are continuous. Therefore, the coefficient of determination is used. Credit scoring is a binary classification problem. For this reason, the area under the ROC curve score (AUROC) is used.

3.6.1 Coefficient of Determination

The coefficient of determination, also known as R-squared or R^2, is the correlation between the predicted and actual values. It indicates the proportion of variance in a continuous dependent variable explained by a model's predictions (Draper and Smith, 1998). The sum of squares of residuals is the sum of squares of the differences between the predicted and the actual values, where $\hat{y_i}$ is a predicted value:

$$SS_{res} = \sum_i (y_i - \hat{y_i})^2. \tag{3.33}$$

The mean of the actual values of the dependent variable is defined as follows, where n is the number of samples:

$$\bar{y} = \frac{1}{n} \sum_{i=1}^{n} y_i \tag{3.34}$$

The total sum of squares is the sum of squares of differences of the actual values and the mean. Also, variance can be deducted from it by dividing by the sample set

size. Its formula is as follows:

$$SS_{tot} = \sum_i (y_i - \bar{y}_i)^2. \tag{3.35}$$

Then, the R^2 is defined as:

$$R^2 = 1 - \frac{SS_{res}}{SS_{tot}}. \tag{3.36}$$

For a random predictor, the expected R^2 value is zero, which indicates none of the variance is explained by predictions. For a perfect predictor, the residual value $(y_i - \hat{y}_i)$ for all samples is expected to be zero. Therefore, $SS_{res} = 0$, and $R^2 = 1$ are expected. A mean predictor outputs the mean $\bar{y}_i$ for all predicted samples. R^2 might take negative values as well for worse than mean predictors.

3.6.2 Reciever Operating Characteristic

A ROC curve indicates a true-positive rate or sensitivity versus a false-positive rate or $1-$Specificity for every threshold value of a binary classification model's outputs. Sensitivity and specificity are previously defined in Equations 2.5 and 2.6.

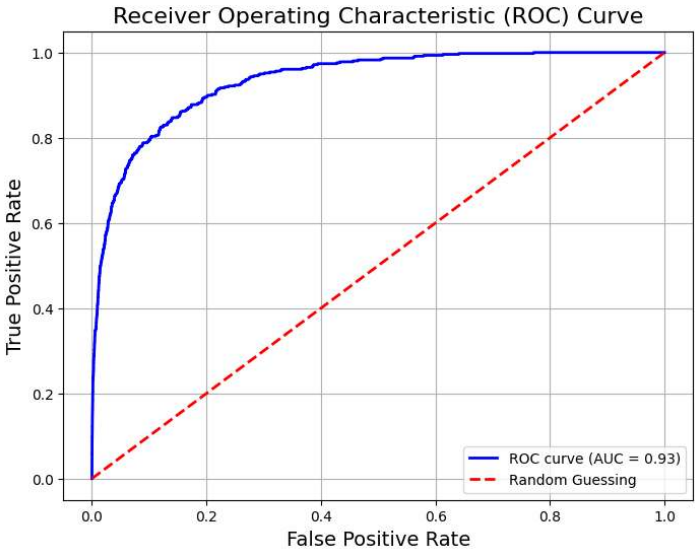


Figure 3.7: An example ROC curve of a credit scoring model with 93% AUROC score on the training set.

Typically, classification models generate class probabilities as output. These probabilities are converted into integer labels $\{-1, +1\}$ through a threshold. Entries with a probability greater than the threshold are marked as positives, and lower probability entries are negatives. ROC curve allows one to examine the trade-off between true-positive and false-positive rates for all threshold values (Mandrekar, 2010). An example ROC curve from this Thesis can be seen in Figure 3.7.

AUROC provides an aggregate description of a model's overall discriminatory power. It goes through every possible threshold value and converts performance at every threshold into a single metric. It takes a maximum value of 1 where a model perfectly distinguishes between the two classes, a minimum of 0 for worst, and 0.5 for indifferent models. As stated by Dastile et al. (2020), it is a frequently used metric in credit scoring. It might be interchangeably used with the Gini coefficient according to Equation 3.37, a popular metric for measuring income disparity between people (Hand and Till, 2001).

$$\text{Gini} = 2 \times \text{AUROC} - 1. \tag{3.37}$$

Chapter 4

GRAPH THEORY

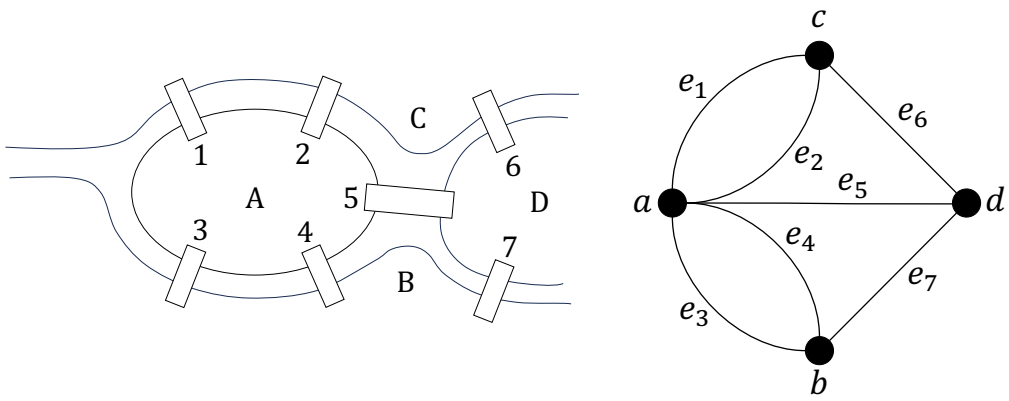
In this chapter, a graph theory review is fulfilled to the extent it is used in this thesis. First, it starts with a generic overview of what is a graph, graph types, and basic graph terminology. Second, a graph representation learning section introduces traditional and modern techniques. Third, another section describes graph neural networks and popular algorithms. Finally, a section conveys details about the GraphSAGE algorithm, basically its sampling, aggregation, loss, and optimization functions.

4.1 Overview

This subsection is written mainly using the work of West (2002). The Pregel River splits the city of Königsberg in Prussia. The separate lands and islands were connected by seven bridges at the time, shown in Figure 4.1a. It was wondered if walking around the city by only passing each bridge exactly once was possible. Although it might be hard to figure out the solution with a basic representation as in Figure 4.1a, a graph representation as in Figure 4.1b makes it easier to see that it is not possible. Leonhard Euler's paper, which argues that there is no viable solution to the Königsberg bridge problem, is accepted as the first paper on graph theory.

A graph G is a collection of a vertex (or node) set $V(G)$, a set of edges $E(G)$ that are connecting pairs of two vertices with each other. Figure 4.1b shows a graph representation where the vertex set is $V(G) = \{a, b, c, d\}$ and the edge set connecting them $E(G) = \{e_1, e_2, e_3, e_4, e_5, e_6, e_7\}$.

Other typical graph problems include finding the fastest route between two cities, ordering a traveling salesman's destinations to minimize travel time, scheduling a sports league into a tight time, and optimizing a computer chip design.



(a) (b)

Figure 4.1: The lands (in letters) and the bridges (in numbers) of the Königsberg city with (a) a basic representation (b) a graph representation.

In this part, some definitions related to the theoretical side of the thesis will be provided. A loop is an edge with the same endpoints. Multiple edges are a set of edges with the same pair of endpoints. A simple graph is a graph without loops or multiple edges. A path is a simple graph where each edge pair shares a different endpoint so that they look consecutive. In a connected graph, each vertex pair is part of a path; otherwise, it is disconnected.

A weighted graph is a form of a graph where a weight $w(e)$ is assigned to each edge. These weights might be used to indicate an edge's costs, distances, and importance. These are initially used to solve the shortest path and traveling salesman problems (Mathew et al., 2023).

In addition to the edge weights and directions additional node and edge features might be included in a graph. For instance, in a social network, people's ages, addresses, likes, and posts might be used as node features. Also, they might have an edge type as being friends, following or blocking each other (Gong and Cheng, 2019). Additionally, common interests might be added as edge features or graph centrality measures as node features.

A walk is a list of vertices and edges $v_0, e_1, v_1, \ldots, e_k, v_k$ such that, for $1 \leq i \leq k$ an edge e_i is between vertices v_{i-1} and v_i. For instance, a citizen's walk in the city of

Königsberg may also be defined as a walk over the city graph. The number of edges in a walk indicates its length. A random walk is a stochastic process indicating a path that starts from a certain node and proceeds by choosing another neighbor at each step. In an unweighted graph, the neighbors are chosen with equal probabilities. In a weighted graph, neighbor probabilities are weighted according to edge weights. A random walk is also controlled by two additional parameters: return parameter p and in-out parameter q. The parameter p sets the likelihood of revisiting an already visited node during the random walk generation. The parameter q has control over the "inward" and "outward" nodes. If $q > 1$, then the walk is biased towards the immediate neighbors of the starting node, therefore approximating to a breadth-first search (Grover and Leskovec, 2016).

A graph H is said to be a subgraph of graph G if $V(H) \subseteq V(G)$ and $E(H) \subseteq E(G)$. Then, it is said that "G contains H" and indicated as $H \subseteq G$.

A directed graph or digraph has edges with directions, pointing to a head vertex from its tail vertex. In addition to elements that define an undirected graph, they have a function that indicates an edge's head and tail vertices. An underlying graph of a directed graph might be deducted by treating edge endpoints as an unordered pair, as shown in Figure 4.2. If an underlying graph of a directed graph is connected, it is said to be weakly connected. If, for each pair of ordered vertices u, v of a directed graph, a path exists from u to v, then it is strongly connected. The directed graph D in Figure 4.2 is strongly connected because a path exists for every vertex pair.

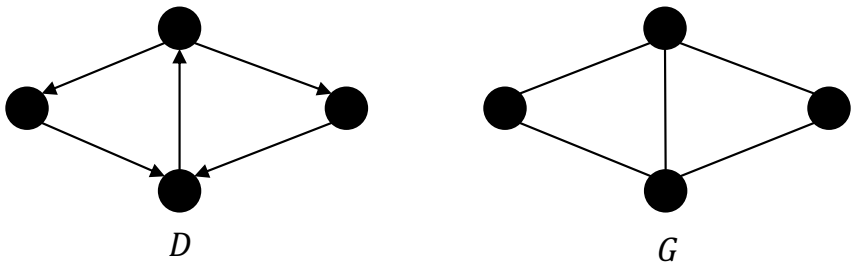


Figure 4.2: A directed graph D on the left, and its underlying graph G on the right.

4.2 Graph Representation Learning

Wu et al. (2022) constitutes a useful resource for examining graph neural networks, which also covers graph representation learning. For this reason, it served as a foundation for subsections covering these subjects.

A representation is the mathematical depiction of input points derived from the feature sets. A machine learning project's success heavily depends on the quality of these representations. Traditionally, representations are embedded via the efforts of people with domain expertise. For instance, if someone's change ratio in indebtedness is to be used, an expert should define it mathematically using historical debt records. Such efforts to create a new useful feature space are called feature engineering. Representation learning aims to automate such procedures by extracting useful but minimal information using algorithms. It aims to reduce the intense effort experts put into feature engineering, expand a probably incomplete representation set, and create an unbiased feature set.

Deep learning is a machine learning technique that uses multiple layers of artificial neural networks. This thesis will mostly concern deep learning-based representation learning on graphs. The deep learning mostly falls under three learning types. Supervised learning is where the true labels of input points are given, and the final layer before the output of a network trained using these labels constitutes a representation space. These labels are non-existent in unsupervised learning, and some pre-tasks might be defined to make it similar to supervised. Transfer learning uses prior knowledge from an initial machine learning procedure to boost performance on another similar task.

Representation learning might be applied to different fields, such as natural language and image processing, network analysis, and speech recognition. Traditional feature engineering on network data mostly focuses on predefined graph-level features such as diameter, average path length, clustering coefficient, node-level features degree and centrality, and subgraph-level features as frequent subgraphs and motifs. Graph representation learning aims to learn low-dimension latent network representations while preserving network topology and vertex features.

Network representation learning faces several challenges that shape the main theme in the research as well. The first challenge is the high computational complexity of adapting mostly iterative or combinatorial algorithms to large-scale networks. The second is the low parallelizability of networks because the highly interconnected structure of vertices creates a high communication cost. The third is being unable to apply machine learning algorithms because they assume independent samples, but a network is about vertices being dependent on each other.

The definition of goodness of a representation depends on the task. For a prediction task, it is capturing the minimum information that is enough for the correct prediction of true labels. To ensure learning a well-performing representation, graph representation learning (GRL) has two goals. First, it is important to be able to reconstruct the original input space from the learned representations. In this representation, vertices with strong connections should have smaller distances to each other. The second goal is to successfully support graph inference tasks, such as node classification, link prediction, or identification of centric nodes.

After the representations are created, traditional machine learning tasks might be applied to any downstream task. GRL might be classified into three subcategories: traditional and modern graph embedding and graph neural networks.

4.2.1 Traditional Graph Embedding

Traditional graph embedding methods emerged as dimensionality reduction techniques. Therefore, they are mostly focused on graph reconstruction. For example, Tenenbaum et al. (2000) has used k-NN to construct neighborhoods and then calculated the shortest paths between nodes. Multidimensional scaling is applied to the distance matrix to learn coordinates as low-dimension representations. However, such efforts within traditional graph embedding assume a well-defined proximity of vertices. This assumption falls short of resolving the complex nature of modern real-world networks of social networks, e-commerce, and biology, where proximity is not well-defined.

4.2.2 Modern Graph Embedding

In contrast to traditional graph embedding techniques, modern graph embedding is mostly focused on graph inference. They consider a richer feature space to support the graph inference from modern complex networks. They might be classified into three categories: (1) structure and property preserving GRL, (2) GRL with side information, and (3) advanced information preserving GRL.

Structure and Property Preserving Graph Representation Learning

Graph structure and property are the two most important components contributing to graph inference. Therefore, a successful GRL technique should learn intrinsic graph properties while preserving structure.

Structure Preserving Graph Representation Learning

GRL's most referred graph structures are neighborhood structures, communities, and high-order node proximity. The way related words follow each other is similar to proximate nodes following each other in a random walk. Therefore, to maximize the probability of a neighbor's taking place in a random walk, DeepWalk used a technique originally used for word embeddings in natural language processing, Skip-Gram (Perozzi et al., 2014; Mikolov et al., 2013). Node2Vec adds sampling bias for changing search strategies of random walk generators from breadth-first to depth-first or somewhere in between, which aims to learn the richer structure of neighborhoods by not assuming linearity as in Skip-Gram (Grover and Leskovec, 2016). Other well-known examples include large-scale information network embedding (LINE) and structural deep network embedding (SDNE) (Tang et al., 2015; Wang et al., 2016).

Property Preserving Graph Representation Learning

Most of the property-preserving GRL is focused on transitivity in graphs and structural balance in signed graphs. Transitivity suggests that if nodes a, b, and a, c are similar, then b, c should also be similar. However, this might not always be true for the graphs. For instance, a person is connected to friends and family in a social network, but they are probably very different. Asymmetric transitivity

suggests that if there are edges from node a to node b, and from b to c in a directed graph, another edge is likely to exist from a to c rather than from c to a.

Network properties build the network's formation and evolution, and property-preserving GRL tries to distill such features by studying the aforementioned theories. Although this research area is relatively new, it shows promise due to its innovative aspects.

Graph Representation Learning with Side Information

Besides graph structure, side information is another knowledge source for graphs. This information can be in node content and node and edge types in heterogeneous graphs. The node content may consist of node labels, attributes, and descriptions. Combining the node content with topology and unifying different types of nodes and edges remain research challenges. Yang et al. (2015) proposed a text-associated DeepWalk, which adapts text features with matrix factorization (Perozzi et al., 2014). It is claimed that their work can integrate the textual content within a research paper or Wikipedia article with their corresponding citation and link networks.

Advanced Information Preserving Graph Representation Learning

Advanced information refers to the (pseudo) supervised information related to a task. This GRL category consists of learning network structure representations and connecting these representations with the target task phases. One important use case is the information diffusion in social networks (Guille et al., 2013). Another one is anomaly detection, mostly used for security and detecting fraudulent actions (Akoglu et al., 2015). The third is graph alignment, finding correspondences or mapping the common nodes of two or more graphs. Its usage includes protein-protein interaction network analysis, cross-lingual document alignment, and alignment of social networks.

4.3 Graph Neural Networks

ANNs were previously introduced in Section 3.5. Although the rich variety in ANN algorithms, they are mostly inapplicable to the graph data as briefly mentioned in Section 4.2 (Wu et al., 2022). To briefly recall, most machine learning algorithms assume that the samples can be represented by independent vectors. However, graph nodes are bound to each other by edges. Also, they do not possess an orderly infrastructure as pixels in an image, words in a sentence, or periods in a time series. Therefore, they do not conform to these algorithms' expectations of inputs with fixed sizes and structures. Adding the need for capturing the node and edge relationships on top of these graph neural networks emerged.

The core of the GNNs is to update the representations iteratively while combining a node's own representations with its neighbors'. Therefore, each network layer has two main functionalities. One is to aggregate information from the neighbors; the other is to update the representations by combining these aggregations with their own representations (Wu et al., 2022).

One well-known GNN architecture is the graph convolutional network (GCN) (Kipf and Welling, 2016a). Inspired by CNN, it uses a localized first-order approximation of spectral graph convolutions. GCN aggregates the neighboring nodes by the weight of their edges, then combines the summation of aggregations normalized by each node's degree. This technique scales linearly in the number of graph edges.

Another important type of GNN is graph attention network (GAT), which is based on the idea that the importance of the features might not be determined by the edge weights (Velickovic et al., 2017). Therefore, it suggests learning these importance parameters automatically through masked self-attention layers. It is parallelizable across node-neighbor pairs, though it might suffer from sparse networks. Once trained, inductive learning may be incorporated into unseen graphs. Its computational complexity level is similar to that of GCNs.

Another GNN algorithm, GraphSAGE, is used in this thesis. It can be trained with fewer nodes as mini-batches of sampled subgraphs rather than the whole graphs as the previously defined GNNs do. As stated in Wu et al. (2023), this leads to less

memory usage. More details about the GraphSAGE algorithm will be provided in the following sections.

4.4 GraphSAGE

GraphSAGE (short for sample and aggregate) is an inductive GRL algorithm that aims to learn generalizable embedding functions starting from node features (node profile, text attributes, etc.) and integrate graph structure (i.e., node degrees) (Hamilton et al., 2017). Therefore, it has an inductive nature that can expand upon the unseen nodes, whereas the GCN algorithm is transductive. Inductiveness is important for expanding networks such as banking customers' networks.

As opposed to matrix factorization-based methods, GraphSAGE does not try to learn a fixed node embedding vector for each node. Instead, it tries to learn aggregator functions to embed local neighborhood features. Depending on the number of aggregation layers set for the algorithm, each aggregator function incorporates features from different neighborhood levels.

4.4.1 Sampling

GraphSAGE uses sampled subgraphs for each node during training. This requires setting a number for hops-neighborhood K, which is also the number of the aggregator layers. Also, the number of neighbors at each neighborhood level $S_1, \ldots, S_K$ must be set initially. The effective range of high performance parameters is $K = 2$ and $S_1 \cdot S_2 \leq 500$. An example of the sampling strategy and construction of the computational graphs is provided in Figure 4.3 (Leskovec, 2023). Here, the aggregator functions are depicted as blue boxes.

4.4.2 Aggregation

K aggregator functions AGGREGATOR_k aggregate neighborhood information into $h^k_{\mathcal{N}(v)}$ and concatenate these with previous layer output h^{k-1}_v, then embed these with weights W^k together with non-linearity σ to construct the k-th layer output h^v_k, where $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, $v \in \mathcal{V}$, $\mathcal{N}(v)$ is the sampled neighborhood for node v, $\forall k \in$

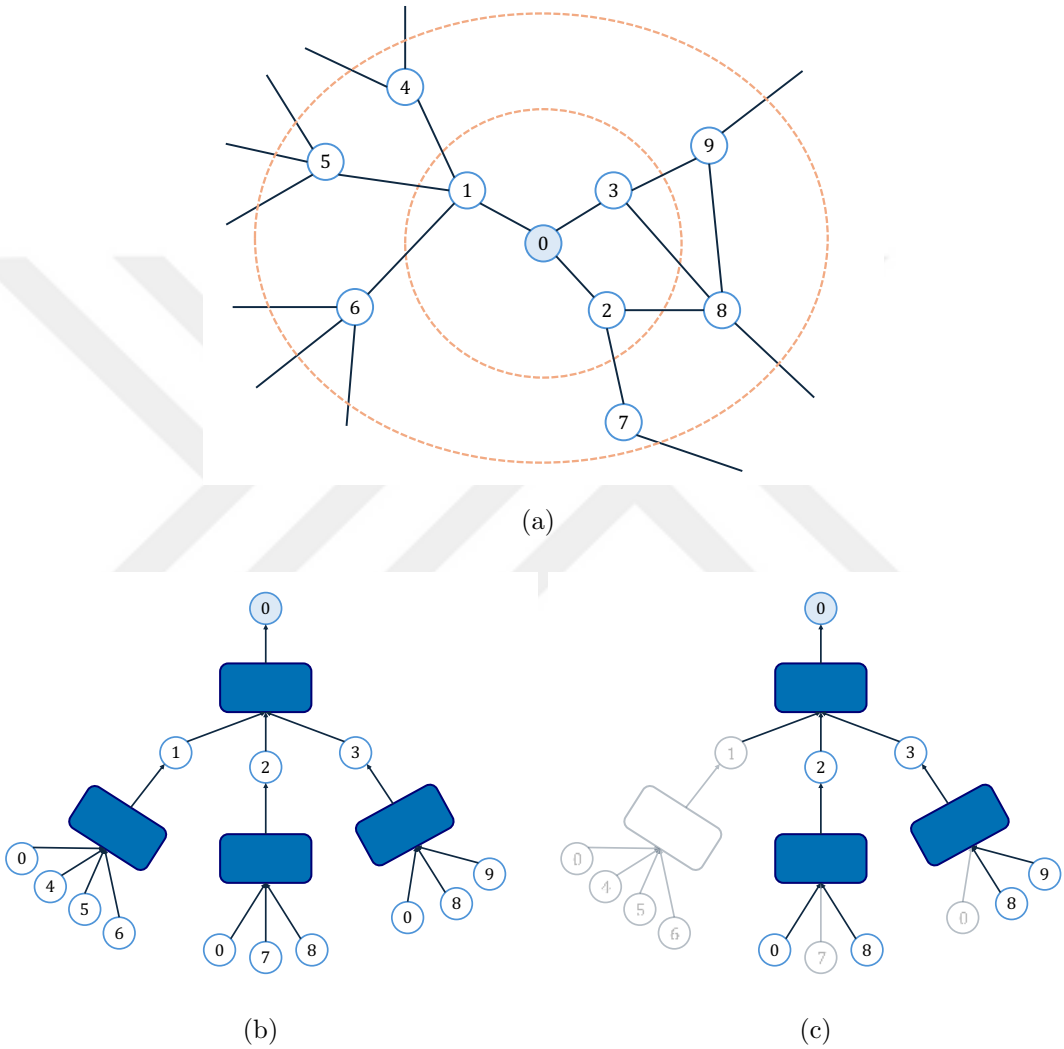


Figure 4.3: Constructing a sampled computational graph with $S_1 = 2, S_2 = 2$ neighbors within $K = 2$ hop-neighborhood: (a) A sample 2-hop neighborhood of a source node 0, (b) the whole computational graph (c) sampled computational graph.

$\{1, \ldots, K\}$ is the number of aggregator layers. The initial aggregation layer starts using node features x_v as h_v^0. At the end of each layer, the embedding vector h_v^k is Euclidean norm (ℓ_2) normalized as $h_v^k / ||h_v^k||_2$.

The mean aggregator takes the elementwise mean of the concatenated vectors h_v^{k-1} and h_u^k where $u \in \mathcal{N}(v)$.

$$h_v^k \leftarrow \sigma\left(W \cdot \text{MEAN}\left(\{h_v^{k-1}\} \cup \{h_u^{k-1}, \forall u \in \mathcal{N}(v)\}\right)\right) \tag{4.1}$$

Pooling aggregators are more complex, where each neighborhood vector h_u^k is fed through a fully connected neural network independently. Then, an elementwise mean or max-pooling operation is applied.

$$\text{AGGREGATE}_k^{\text{pool}} = \max\left(\sigma\left(W_{\text{pool}} h_{u_i}^k + b\right), \forall u_i \in \mathcal{N}(v)\right) \tag{4.2}$$

Another aggregator function used is the attentional aggregator from the GAT architecture (Velickovic et al., 2017). Similarly, it starts with node features h, applies a shared linear transformation through weight matrix W, and passes through a shared attentional mechanism a, a single-layer feed-forward neural network, to calculate attention coefficients

$$e_{ij} = a(Wh_i, Wh_j) \tag{4.3}$$

where i and j are nodes in a graph and e_{ij} is the importance of node j to neighbor node i. Then, the coefficients are normalized using the softmax function to ensure comparability across the whole graph:

$$\alpha_{ij} = \text{softmax}_j(e_{ij}) = \frac{\exp(e_{ij})}{\sum_{k \in \mathcal{N}_i} \exp(e_{ik})} \tag{4.4}$$

After extending the attention mechanism as K multi-head and applying another non-linearity σ, the features are concatenated to construct the final output:

$$h_i' = \Big\Vert_{k=1}^{K} \sigma\left(\sum_{j \in \mathcal{N}_i} \alpha_{ij}^k W^k h_j\right) \tag{4.5}$$

where $\Vert$ represents the concatenation operations.

4.4.3 Loss and Optimization Functions

The original paper uses an unsupervised loss function that aims for neighboring nodes to have similar and disparate nodes distinguishing representations.

$$J_G(z_u) = -log(\sigma(z_u^\top z_v)) - Q \cdot \mathbb{E}_{v_n \sim P_n(v)} \log(\sigma(-z_u^\top z_{v_n})) \tag{4.6}$$

In this equation, u and v are the co-occurring nodes from a fixed-length random walk, whereas v_n is a sampled node using the negative sampling function P_n, and Q is the number of negative samples. In our application using `stellargraph`, the positive and negative samples are similarly simulated in a random walk, but a binary cross-entropy is used.

The paper suggested using stochastic gradient descent (SGD) to optimize the loss function during backpropagation. Our choice has been the Adam (for adaptive moment estimation) optimizer (Kingma and Ba, 2014). Adam optimizer is a modification to the SGD where the learning rate is adjusted by a signal-to-noise ratio calculated using first and second moments. In this way, a greater step is taken when there is a strong signal, and a smaller step is taken when the noise increases as the loss converges to the global optimum.

Chapter 5

EXPERIMENTS AND RESULTS

The core purpose of this thesis is to improve a bank's small and medium-sized enterprises (SMEs) behavioral credit scoring model performance by adding graph representation learning embeddings generated using enterprise money transfer information on top of the existing model scores. For this reason, a series of experiments with graph representation learning and machine learning algorithms in various settings are run to achieve the best possible performance. This chapter will convey a detailed description of the experiment framework. First, a detailed overview of the data sets will be presented. Second, the preprocessing steps will be shared. Then, the methods and results in these stages will be explained. Finally, the chapter will discuss the outcomes of the performance analyses, highlighting the significant contributions of the proposed model.

5.1 Data Set

In this thesis, QNB Finansbank's corporate customer data set is used. The data set included both the money transfer and descriptive customer features. The transfer data for the edge weights definition of the graph covered one year between 11/2021 and 10/2022. Other time period variations are added as node features. The descriptive features are a customer snapshot at the interval end, 10/2022. The data set construction consists of two phases: first, creating the initial data set for learning node embeddings through graph representation learning, and second, concatenating these features with segment and existing credit scoring model score for the new credit scoring model.

5.1.1 Graph Representation Learning Data Set

For the graph representation learning phase, a graph representing companies and their relationships is constructed. In this graph, companies are represented as nodes and the sum of the pairwise transactions during the time interval as edges. Generally speaking, adding the non-portfolio customers to the graph might be a problem, the ones who have not worked with the bank so far. The reason is that they might use different accounts in different banks with slightly different names. Therefore, one non-portfolio customer might have appeared in multiple nodes with similar names. With a text-similarity-based prior study, such multiple accounts have been merged into a single name and ID beforehand. Then, it has become possible to use these customers as well.

The initial number of edges in the constructed graph was 1.67 million. A company's transfers to itself may indicate salary or similar payments, which is irrelevant for extracting relationships. After their exclusion, the number was down to 1.52 million. To prevent very short random walks from occurring during representation learning, the nodes without an incoming and an outgoing edge and their edges are removed. This removal has to be carried out several times because each removal round creates such new nodes. This removal process must be repeated until the graph is strongly connected. These operations have reduced the number of nodes from about 347,000 to 164,000. The number of edges is diminished to 1.11 million as well.

Companies' descriptive information in the bank is added as node features to enhance the discriminatory power of the GRL algorithm's representations. Table 5.1 provides feature abbreviations and explanations. Some of them are reported as groups of variables, and group sizes are written in parentheses.

5.1.2 Credit Scoring Data Set

A target definition is needed first to train a credit scoring model. As in the target definition of Dastile et al. (2020), customers' inability to repay their debt for more than 90 days became the target definition in the thesis. The target is observed for a

Table 5.1: List of node features used in graph representation learning. A feature group is indicated with the number of features it comprises written in parentheses.

Abbreviation	Explanation
sector	The sector that the company operates in
class	Whether the company has performed any default or not
pc code	Company size segment
credit region	Company's geographical region
proprietorship	Whether the company is an individual proprietorship or not
beh pd	Behavioral credit score from the current model
portfolio flag	Whether the company is a bank customer
risk variables (23)	Limits, risks, other credit risk model scores, etc.
transfer variables (18)	Transfer variables with different variations in incoming/outgoing amount/frequency

year after the end of the input observation period, from 11/2022 to 10/2023. For a company to have a target label, it should be a bank customer with positive ongoing indebtedness.

Previously defaulted customers are easy to figure out for a model, and they deteriorate a model's performance. Therefore, they should be removed from the data as well. After filtering the SMEs with positive risk but not with a prior default, the number of customers in the data set is down to around 49,500. The default ratio is the number of customers who satisfy target conditions compared to the whole sample set size. The number of customers within each segment and their default ratios are given in Table 5.2.

Ideally, two separate models for both of the small and medium-sized enterprise segments might have been developed. However, the number of samples and the default ratio are low for the medium-sized segment, which will not constitute a sample set big enough for a successful machine-learning task. For this reason, a

Table 5.2: Segment sample sizes and default ratios for credit scoring data set.

Segment	Number of Samples	Default Ratio
Small-Sized Enterprises	35,254	1.05%
Medium-Sized Enterprises	14,247	0.60%
Total	**49,501**	**0.92%**

single model encapsulating both segments is developed.

5.2 Preprocessing

Data preprocessing is another fundamental process in machine learning that prepares data for the actual task. It aims to enhance accuracy, robustness, and data quality through data cleaning, integration, transformation, and reduction. A general introduction to data preprocessing in credit scoring exists in Section 2.1. In this thesis, inflation adjustment, categorical variables encoding, and imputation are applied as data preprocessing.

5.2.1 Inflation Adjustment

Since the Turkish Lira (TRY) has been subject to high inflation lately, the transaction amounts must be mapped to a similar scale for every period. Several different indicators for an inflation adjustment to the Turkish Lira exist. Turkish Statistical Institute (TURKSTAT), Istanbul Chamber of Commerce (ICOC) Producer Price Indexes (PPI), and US Dollars to TRY currency ratio (USD TRY) are evaluated on the average customer transaction amounts.

The compared inflation-adjustment indicators in Figure 5.1 mostly track each other through time. However, ICOC PPI yields more intermediate values than others, with an overall lower variance. Therefore, it is chosen for the adjustment.

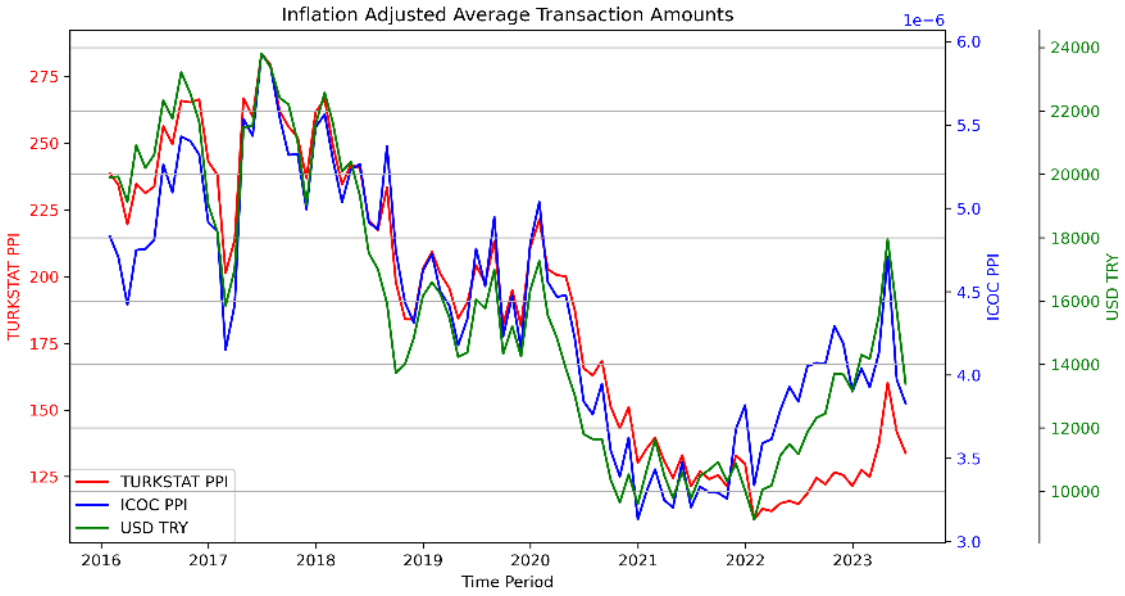


Figure 5.1: Inflation adjustment results of the transaction means according to TURKSTAT and ICOC PPIs and USDTRY values. Each indicator's scale is denoted on the sides.

5.2.2 Encoding Categorical Variables

Categorical variables consist of non-numerical values named as categories, in the form of $\{a, b, c, d, e, \ldots\}$ for instance. Some examples of categorical variables from our data set are the credit region, sector, and segment of the customers. To be able to be processed mathematically, they are converted to multiple binary vectors using one-hot encoding. This encoding converts N distinct categorical values in a vector into N binary vectors for each value. Any entity's encoded value is 1 for its categorical value vector and 0 for any other encoded vector. A simple example of encoding is given in Equation 5.1, where a is an attribute vector with categorical values, $\{e_1, e_2, e_3, e_4, e_5\}$ are set of entries, a_i is an encoded attribute vector indicating

if an entry has value i in its original attribute vector a.

$$\begin{array}{c} \\ e_1 \\ e_2 \\ e_3 \\ e_4 \\ e_5 \end{array} \begin{array}{c} a \\ \left[\begin{array}{c} a \\ b \\ c \\ d \\ e \end{array}\right] \end{array} = \begin{array}{c} \\ e_1 \\ e_2 \\ e_3 \\ e_4 \\ e_5 \end{array} \begin{array}{c} \begin{array}{ccccc} a_a & a_b & a_c & a_d & a_e \end{array} \\ \left[\begin{array}{ccccc} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{array}\right] \end{array} \tag{5.1}$$

Additionally, distinct values with fewer than a 100 frequency are grouped as "other" in an additional vector.

5.2.3 Imputation

Null values indicate an entry has no value for that feature vector. A few columns with null values are imputed using machine learning model predictions developed separately for each column with decision tree (DT) and random forest (RF) using `scikit-learn` package (Pedregosa et al., 2011). Simpler columns that are common with bank customer and non-customer entities are used as input features. In addition, variables that mostly have zero values for entities with null values are excluded. Selected input variables consist of "sector", "pc code," "credit region," "class," "proprietorship," and "transfer variables," as their descriptions can be recalled from Table 5.1. Imputation models are trained with entries without any null input values.

k-fold cross-validation is used to choose the best model. It splits the data set into k equal parts and learns models k times with different train and validation data sets and the same hyperparameters. A different part is set as validation for each run, and the rest is set as training. Therefore, a model's validity is assured on the whole data set. The number of folds k is set as 3 for the imputation.

Random search is utilized for hyperparameter optimization. It generates a random hyperparameter set before every model fit from given intervals of hyperparameters. The hyperparameter intervals for the imputation are given in Table 5.3. Hyperparameter explanations are taken from Pedregosa et al. (2011). The continu-

ous search space intervals are denoted with floating numbers.

Table 5.3: Hyperparameter search space of decision tree and random forest for imputation.

		Search Space	
Hyperparameter	Explanation	Decision Tree	Random Forest
max_depth	The maximum depth of tree	[3, 20]	[5, 50]
min_samples_split	The minimum number of samples required to split an internal node	[2, 20]	[2, 20]
min_samples_leaf	The minimum number of samples required to be at a leaf node	[1, 20]	[1, 20]
max_features	The fraction of features to consider when looking for the best split	[0.05, 1.0]	[0.25, 1.0]
n_estimators	The number of trees in the forest	-	[10, 200]
bootstrap	Whether bootstrap samples are used when building trees or not	-	[True, False]

40 DTs and 20 RFs with different hyperparameters are generated for each feature. Imputed variables' null ratios and fitted models' R^2 scores are provided in the Table 5.4. If any of these features has not been explicitly indicated in Table 5.1, it is part of the "risk variables" feature group. All the selected models are built with RFs, as indicated in bold. Therefore, DTs are acted as benchmarks.

An independent variable's being null might be meaningful for the machine learning models. For this reason, binary vectors are added, indicating if a variable of an entry has been imputed. With the addition of these and one-hot encoded variables, the number of input variables is increased to 111. To prevent any possible dominance of a variable, all input variables are standardized by subtracting its average and dividing by unit variance as in Equation 5.2.

$$x' = \frac{x - \bar{x}}{\sigma} \tag{5.2}$$

Table 5.4: Null ratios and prediction powers of machine learning models as the coefficient of determination for imputed variables.

		R^2	
Variable	Null Ratio	Decision Tree	Random Forest
Term profit of customer	4.8%	5.2%	**8.9%**
All total limit	48.2%	32.0%	**40.4%**
All total risk	48.2%	24.4%	**32.1%**
All cash risk	48.2%	25.4%	**26.6%**
Number of banks that customer is working with	48.2%	43.5%	**48.8%**
Rating probability of default	11.8%	24.4%	**27.8%**
Behavioral probability of default	44.9%	94.1%	**94.7%**

5.3 Experiments

This subsection will share the common setting for GRL first, then for credit scoring. Afterward, results with different GRL settings will be shown.

5.3.1 Experimental Settings for Graph Representation Learning

For graph representation learning, a directed graph with money transfer amounts as edge weights is defined. A weighted, unsupervised random walk sampler is added afterward. A weighted and directed GraphSAGE link generator is used to generate training subgraphs. A directed GraphSAGE model with bias, 0.3 dropout, ℓ_2-normalization is learned. Dropout randomly removes a "neuron"s weight and incoming and outgoing connections during training with a probability p, here 0.3. This prevents neural networks from overfitting (Srivastava et al., 2014). ℓ_2-Normalization is used in GCN embeddings to prevent the nodes from shrinking too much in the representation space that they become indistinguishable (Yang et al., 2020).

Different activation functions are used: ReLU, leaky ReLU, hyperbolic tangent

(tanh), sigmoid, and linear. The sigmoid activation function is used in the final output layer. It is a monotonic, differentiable activation function that can turn any input into a probability distribution, as previously defined in Equation 3.9. These properties make it a frequently used final layer activation function in neural networks.

In our experiments, the initial learning rate η_0 is set to 0.01 and reduced at each epoch end using an exponential decay with a 0.5 decay rate d. Therefore, the learning rate at epoch t, η_t is expressed in Equation 5.3.

$$\eta_t = \eta_0 e^{-dt} \tag{5.3}$$

Binary cross-entropy loss is used to track model performance. Early stopping is a simple regularization method to prevent overfitting that stops further training according to a matching criterion. A minimum improvement of 0.001 in validation loss and 4 epochs of patience are used for early stopping. GraphSAGE framework is applied using `stellargraph` library (CSIRO's Data61, 2018).

5.3.2 Experimental Settings for Credit Scoring

For credit scoring, a framework similar to that in the imputation phase is used for all graph embedding settings. Compared to the imputation phase, the number of model runs is increased, and extreme gradient boosting (XGB) and logistic regression (LR) are used instead of RF. Number of runs are set to 40 for DT, 500 for XGB, and 100 for LR. A test set of 20% of the samples is separated. Again, a random search for hyperparameter optimization is applied, and the number of folds for k-fold cross-validation is increased to 5. The optimum hyperparameter search space for DT does not differ from the imputation phase, as indicated in Table 5.3. The search spaces for XGB and LR are provided in Table 5.5 and 5.6, respectively. Hyperparameter explanations are again taken from the `scikit-learn` (Pedregosa et al., 2011). Floating numbers indicate continuous search space intervals.

Table 5.5: Hyperparameter search space of extreme gradient boosting for credit scoring.

Hyperparameter	Explanation	Search Space
n_estimators	Number of boosting round	[5,200]
learning_rate	Boosting learning rate, "eta"	[10^{-4}, 1.0]
max_depth	Maximum tree depth for base learners	[5, 50]
subsample	Subsample ratio of the training instance	[0.25, 1.0]
colsample_bytree	Subsample ratio of columns when constructing each tree	[0.25, 1.0]
min_child_weight	Minimum sum of instance weight (hessian) needed in a child	[1, 40]
gamma	Minimum loss reduction required to make a further partition on a leaf node of the tree	[0.0, 1.0]
reg_alpha	ℓ_1 regularization term on weights	[0.0, 1.0]
reg_lambda	ℓ_2 regularization term on weights	[0.0, 1.0]

5.3.3 Results

Results for credit scoring models using GRL embedding features are indicated in Table 5.7 as AUROC scores. Looking at this table, if a graph has directed edges, a directed GraphSAGE is constructed, doubling the number of parameters of the network compared to an undirected graph. The reason is that different weights are initialized for both "in" and "out" edges. Therefore, almost doubling the computational needs as well. If a graph has weighted edges, weight-biased random walks are simulated. Walk lengths refer to the lengths of the random walks.

The probability "p" defines the likelihood of returning to the source node, while "q" represents the probability of moving away from the source node during a random walk simulation. The number of samples denotes the count of first and second-hop neighbors in a subgraph, and it is set the same for both incoming and outgoing edges in directed graphs.

Table 5.6: Hyperparameter search space of logistic regression for credit scoring.

Hyperparameter	Explanation	Search Space
C	Inverse of regularization strength	$[10^{-4}, 10^{4}]$
fit_intercept	If a constant should be added to the decision function	[True, False]
solver	Algorithm to use in the optimization problem	[“newton-cg”, “lbfgs”, “liblinear”, “sag”, “saga”]
max_iter	Maximum number of iterations taken for the solvers to converge	[1000, 5000]
class_weight	Weights associated with the classes	[“balanced”, None]

The layer sizes correspond to the hidden layer sizes of a neural network with two hidden layers, where each layer has the stated size. Aggregators indicate the types of GraphSAGE neighborhood aggregators used in the models. If there are multiple aggregators, it means that multiple model embeddings constructed using these aggregators are concatenated. Consequently, the size of the last layer also represents the GRL's embedding size. Embedding sizes are multiplied by the number of aggregators to reach the total embedding sizes. The activations are the activation functions, and initializations are the weight initializations of neural networks as previously described in Section 3.5. The existing model had 89.20% AUROC on the train set and 86.53% AUROC on the test.

Table 5.7: AUROC scores of credit scoring using graph representation learning embeddings. The selected model is indicated with (*).

									AUROC Improvement %					
									Decision Tree		XGBoost		Logistic Regression	
Edge Type	Walk Length	Probabilities	Number of Samples	Layer Sizes	Aggregators	Total Embedding Size	Activations	Initialization	Validation	Test	Validation	Test	Validation	Test
Undirected & unweighted	5	p = 1.0, q = 1.0	[10, 5]	[50, 50]	Mean	50	[relu, linear]	Glorot Uniform	-4.46	-4.94	0.23	1.37	-0.29	1.86
Directed & weighted	5	p = 0.5, q = 2.0	[10, 5]	[50, 50]	Mean	50	[relu, linear]	Glorot Uniform	-4.57	-4.68	0.10	2.37	-1.23	2.78
Directed & weighted	5	p = 0.5, q = 2.0	[10, 5]	[128, 128]	Mean	128	[relu, linear]	Glorot Uniform	-4.90	-1.52	0.06	2.51	-1.26	3.34
Directed & weighted	5	p = 0.5, q = 2.0	[20, 10]	[128, 128]	Mean	128	[relu, linear]	Glorot Uniform	-4.67	-1.45	0.27	3.28	-1.50	3.13
Directed & weighted	5	p = 2.0, q = 0.5	[20, 10]	[128, 128]	Mean	128	[relu, linear]	Glorot Uniform	-4.17	-4.25	0.23	1.21	-1.51	2.77
Directed & weighted	7	p = 0.5, q = 2.0	[20, 10]	[128, 128]	Mean	128	[relu, linear]	Glorot Uniform	-4.33	-1.22	-0.13	1.40	-1.52	3.28
Directed & weighted	5	p = 0.5, q = 2.0	[20, 10]	[256, 256]	Mean	256	[relu, linear]	Glorot Uniform	-1.81	-1.47	**0.63**	1.64	-1.43	2.96
Directed & weighted	5	p = 0.5, q = 2.0	[20, 10]	[256, 256]	Mean	256	[leaky relu, leaky relu]	Glorot Uniform	-4.47	-2.56	0.14	1.79	-1.18	**4.50**
Directed & weighted	5	p = 0.5, q = 2.0	[20, 10]	[256, 256]	Mean	256	[relu, relu]	Glorot Uniform	**-1.59**	-0.96	0.43	2.28	-0.44	3.00
Directed & weighted	5	p = 0.5, q = 2.0	[20, 10]	[256, 256]	Attention	256	[relu, relu]	Glorot Uniform	-4.03	-7.84	0.11	0.39	-2.23	-0.84
Directed & weighted	5	p = 0.5, q = 2.0	[20, 10]	[256, 128]	Mean	128	[relu, relu]	Glorot Uniform	-2.85	-0.94	0.31	2.14	**-0.27**	3.01
Directed & weighted	5	p = 0.5, q = 2.0	[20, 10]	[256, 256]	Max Pool	256	[relu, relu]	Glorot Uniform	-1.71	-1.46	0.09	1.31	-1.08	1.99
Directed & weighted	5	p = 0.5, q = 2.0	[20, 20]	[256, 256]	Mean	256	[relu, linear]	Glorot Uniform	-2.53	-1.72	0.09	2.10	-1.46	1.43
Directed & weighted	5	p = 0.5, q = 2.0	[20, 10]	[512, 512]	Mean	512	[relu, linear]	Glorot Uniform	-6.72	-2.27	0.23	2.40	-1.38	3.48
Directed & weighted	5	p = 0.5, q = 2.0	[30, 10]	[256, 256]	Mean	256	[relu, linear]	Glorot Uniform	-4.31	-1.52	0.09	2.79	-1.35	3.12
Directed & weighted	5	p = 0.5, q = 2.0	[30, 10]	[256, 256]	Attention	256	[relu, linear]	Glorot Uniform	-4.11	-1.51	0.12	0.92	-1.31	0.79
Directed & weighted	5	p = 0.5, q = 2.0	[30, 10]	[256, 256]	Mean Pool	256	[relu, linear]	Glorot Uniform	-3.47	-1.56	0.47	2.70	-0.89	3.10
Directed & weighted	5	p = 0.5, q = 2.0	[30, 10]	[256, 256]	Max Pool	256	[relu, linear]	Glorot Uniform	-3.83	-2.39	0.22	1.67	-1.14	2.61
Directed & weighted	5	p = 0.5, q = 2.0	[30, 10]	[256, 256]	Attention & Max Pool	256	[relu, linear]	Glorot Uniform	-7.44	-6.89	0.60	2.36	-1.36	2.50
Directed & weighted	5	p = 0.5, q = 2.0	[30, 10]	[256, 256]	Mean & Attention	512	[relu, linear]	Glorot Uniform	-8.39	-6.72	0.48	3.13	-1.41	2.70
Directed & weighted	5	p = 0.5, q = 2.0	[30, 10]	[256, 256]	Mean & Max Pool	512	[relu, linear]	Glorot Uniform	-7.15	-2.71	0.30	2.53	-1.80	2.62
Directed & weighted	5	p = 0.5, q = 2.0	[30, 10]	[256, 256]	Mean & Mean Pool	512	[relu, linear]	Glorot Uniform	-8.67	-9.68	0.45	2.74	-1.27	3.21
Directed & weighted	5	p = 0.5, q = 2.0	[30, 10]	[256, 256]	Mean & Attention & Max Pooling	768	[relu, linear]	Glorot Uniform	-6.33	**0.55**	**0.62***	**3.70***	-2.40	0.70
Directed & weighted	5	p = 0.5, q = 2.0	[30, 10]	[256, 256]	Mean	256	[relu, linear]	He Normal	-4.66	-1.48	0.27	1.95	-1.14	3.34
Directed & weighted	5	p = 0.5, q = 2.0	[30, 10]	[256, 256]	Mean	256	[relu, linear]	He Uniform	-4.98	-1.51	0.1	2.26	-1.41	3.13
Directed & weighted	5	p = 0.5, q = 2.0	[30, 10]	[256, 256]	Mean	256	[sigmoid, linear]	Glorot Normal	-1.84	-1.43	0.09	2.41	-1.31	2.71
Directed & weighted	5	p = 0.5, q = 2.0	[30, 10]	[256, 256]	Mean	256	[tanh, linear]	Glorot Normal	-5.75	-1.48	0.06	2.80	-1.54	2.84

Table 5.7 shows that there is not any embedding set that improves the validation set scores with logistic regression. Therefore, it is possible to speculate that either the embeddings are not linearly separable in terms of credit scoring or they are only useful on certain sample groups so that only XGBoost can benefit.

Looking at the improvements in both the validation and the test sets, we chose an XGBoost model that is indicated by a (*) in Table 5.7. The selected model brings an AUROC increase of 0.62% on validation and of 3.70% on the test set. It uses concatenated embeddings of three different aggregators: mean, attention, and max pooling. The following section inspects performance improvements in various aspects.

5.4 Performance Analyses

This section will share performance improvements by models' scores percentiles and expected economic contribution according to such analysis. Afterward, performance improvement analysis results regarding incoming and outgoing edge counts and interaction amounts are shown. All the analyses are performed on the test set, which has 9,901 samples and 0.93% default ratio to recall.

5.4.1 Performance Improvements by Model Scores and Economic Contribution

Generally, when a new credit scoring model is developed, the first analysis is to look at both new and old models' score percentile groups and how much default ratio will be saved through the transition to the new model. Table 5.8 provides one such analysis.

Looking at this analysis, the bank might choose to sustain the credit limits of enterprises of the first 90 percentile of the new model. This selected sample set's default ratio would be down from 0.35% to 0.26% by switching from the old model to the new one. To put it in real terms, such a switch reduces the number of defaulters of small-sized enterprises from 27 to 20 and the middle-sized from 4 to 3 in the selected set. For the target period, the small-sized enterprises had 10,136$ and the middle-sized 77,350$ as average default amounts. Therefore, the new model savings

Table 5.8: Old and new credit score models' default ratios regarding their score percentile intervals and new model's improvement on the old one.

Models' Score Percentiles	Old Model Default Ratio	New Model Default Ratio	Net Default Ratio Improvement
[0-10)	0.00%	0.00%	0.00%
[10-20)	0.10%	0.00%	0.10%
[20-30)	0.10%	0.00%	0.10%
[30-40)	0.39%	0.41%	-0.02%
[40-50)	0.42%	0.20%	0.22%
[50-60)	0.10%	0.10%	0.00%
[60-70)	0.31%	0.10%	0.21%
[70-80)	0.83%	0.77%	0.06%
[80-85)	0.63%	0.42%	0.21%
[85-90)	1.11%	1.01%	0.10%
[90-95)	2.03%	3.37%	-1.34%
[95-99)	6.56%	7.73%	-1.17%
[99-100]	28.09%	23.08%	5.01%

on the selected set would be 148,303$ if the bank would freeze the credit limits.

5.4.2 Number of Edges Performances

Here, an enterprise's number of incoming and outgoing edges' impact on the performance is concerned. The number of edges are examined as groups of 1, 2, [3-4], [5-8], [9,∞). Such a choice of edge numbers in groups almost uniformly distributes the number of samples among them. The analysis is performed using the selected model on the test set. Results are shown in Table 5.9.

Looking at the table, it can be seen that performance on enterprises with either one incoming or outgoing edge has improved more than others. The reason might be

Table 5.9: AUROC performance contributions regarding the number of incoming and outgoing edges. Cells indicated with only "-" do not have a target positive sample in their set.

	Number of Incoming Edges					
Number of Outgoing Edges	1	2	[3-4]	[5-8]	[9,∞)	**Weighted Average**
1	7.48%	-3.50%	10.50%	-1.67%	-	5.15%
2	0.09%	-3.98%	-3.24%	-	-	-1.74%
[3-4]	-1.36%	3.11%	3.31%	6.11%	-	2.68%
[5-8]	-	-1.48%	-0.38%	1.86%	-	0.57%
[9,∞)	0.00%	0.00%	-	12.35%	1.97%	3.92%
Weighted Average	5.41%	-0.87%	2.71%	3.81%	2.49%	3.70%

that they initially had a lower AUROC score, 72.51% at their intersection. Another observation might be the worsening of the enterprises with either two incoming or outgoing edges.

5.4.3 Interaction Amount Performances

Similarly, this section investigates model performances according to customers' transfer amounts. Since inflation adjustment already removes the meaning of the amounts, they are grouped by their percentiles as [0-20), [20-40), [40-60), [60-80), and [80-100]. The results can be observed in Table 5.10.

The table shows that the [20-40) percentile groups are improved more. The probable reason is that their intersection has a lower initial AUROC of 66.52%. Another observation here is that although the intersection of [0-20) percentile groups has an initial AUROC of 72.79%, it had a worse performance in the end.

Table 5.10: AUROC performance contributions regarding the transaction amount outgoing from and incoming to a customer.

	Incoming Transaction Amount Percentile					
Outgoing Transaction Amount Percentile	[0-20)	[20-40)	[40-60)	[60-80)	[80-100]	**Weighted Average**
[0-20)	-1.24%	0.51%	-3.94%	-	-13.33%	-1.18%
[20-40)	8.08%	16.63%	-2.42%	4.17%	0.65%	9.19%
[40-60)	0.68%	2.59%	1.61%	3.84%	17.91%	4.03%
[60-80)	0.00%	-	-0.32%	-0.37%	2.60%	0.99%
[80-100]	-	-	3.40%	-0.13%	3.41%	2.78%
Weighted Average	1.23%	7.93%	-0.20%	2.29%	4.49%	3.70%

5.4.4 Credit Score Change Analysis Based on Neighborhood

In this section, the new credit scoring model's adaptation ability to previously defaulted and other risky neighbor information is investigated. They are examined through the "NPL ratio" and "existing credit score," respectively. For this purpose, the percentile difference is calculated as a sample's new model score percentile minus the existing model score percentile. This means the new model sees more risk than the existing model as going to the right of the figures below and vice versa.

Looking at Figure 5.2, the NPL ratio of outgoing neighbors shows a behavior close to a monotonically increasing, and it is more explicit in the weighted version. This means that the new model appointed higher risks to the companies that transfer money more to previously defaulted companies. Incoming edges do not exhibit as much a monotonic behavior.

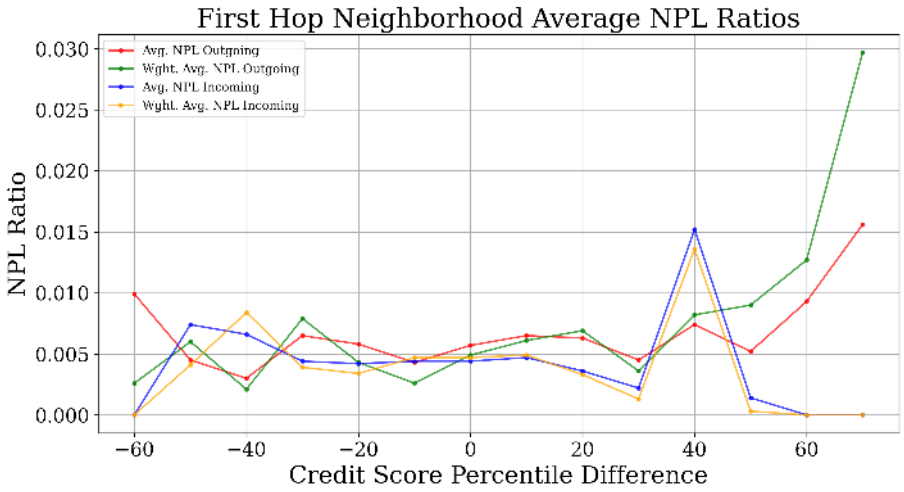


Figure 5.2: Credit score percentile difference between the existing and the new model compared against the (weighted) average NPL ratios of the first hop neighbors on the outgoing/incoming edges.

The average NPL ratio of second hop neighbors does not show much monotonicity as well, except for an increase at the 60 percentile difference as can be seen in

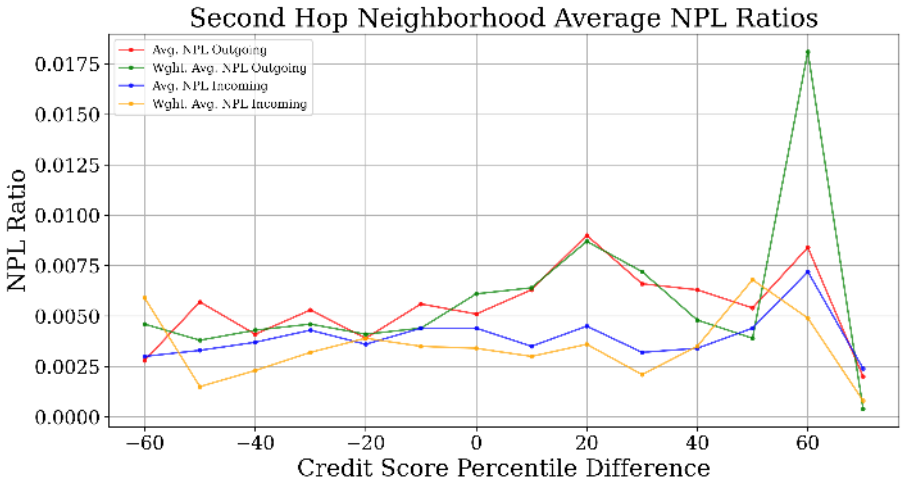


Figure 5.3: Credit score percentile difference between the existing and the new model compared against the (weighted) average NPL ratios of the second hop neighbors on the outgoing/incoming edges.

Figure 5.3.

While performing the same analysis for the existing model's credit scores, previously defaulted or delinquent neighbors are excluded. The reason is that they are already appointed with the highest credit score and already observed in the NPL ratio analysis. A higher score means a higher probability of default to quickly recall. It might be observed in Figure 5.4 that first-hop neighborhood average credit scores show the strongest monotonicity among all. Again, the weighted average of outgoing edge neighbors has the most explicit monotonicity. In addition, there are two unexpected increases in credit scores at 0 and 20 percentile differences.

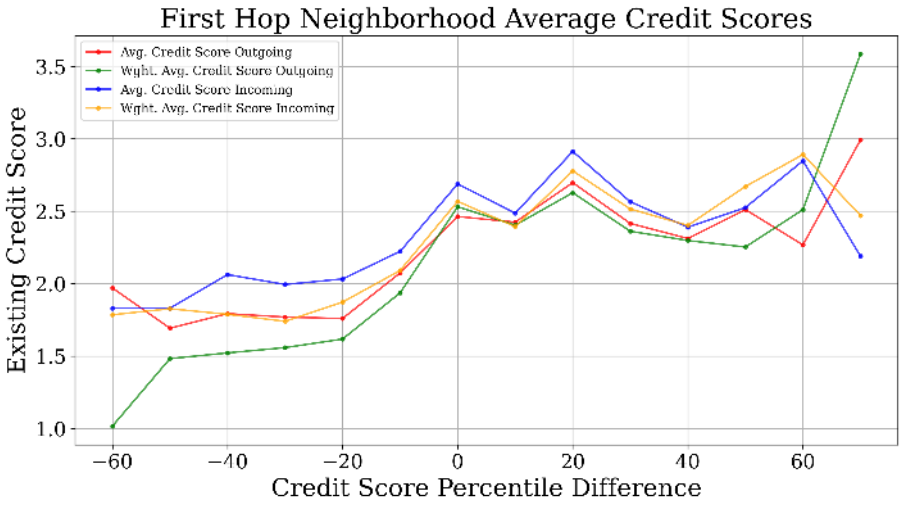


Figure 5.4: Credit score percentile difference between the existing and the new model compared against the (weighted) average existing model credit score of the first hop neighbors on the outgoing/incoming edges.

Figure 5.5 depicts that the second-hop neighborhood average credit scores are weaker monotonicity compared to the first-hop neighborhood, yet still stronger than the second-hop NPL ratios.

The detailed number of samples, target and neighbor NPL ratios, and neighbor existing credit scores are provided in Appendix A.

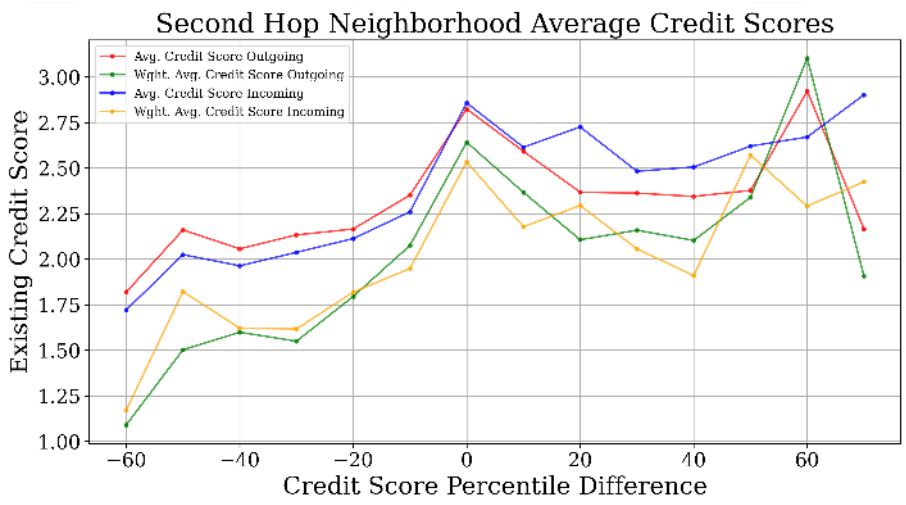


Figure 5.5: Credit score percentile difference between the existing and the new model compared against the (weighted) average existing model credit score of the second hop neighbors on the outgoing/incoming edges.

Chapter 6

CONCLUSION

Credit scoring is one of the most essential problems in banking. In this thesis, a graph neural network algorithm, GraphSAGE, is used to improve the performance of a bank's SME behavior scorecard. This improvement is achieved through incorporating these enterprises' money transfer information into the existing credit score model using GraphSAGE embeddings.

In our experiments, it is seen that concatenating embeddings of different aggregators and combining these embeddings with extreme gradient boosting brought the most improvement. The new credit scoring model has decreased the number of defaulters on the test set. From this, an expected savings amount is calculated.

In addition, the new model brought the most contribution on nodes of the graph with either the least interaction amounts or the least number of edges. These groups had the lowest initial AUROC scores as well. However, groups of nodes with slightly more edges or interaction amounts seemed to have a worse performance in the end, which might be subject to further research.

One important problem with our architecture is that to generate new credit scores, 11 machine learning models are used in total (excluding the existing credit scoring model). Such a practice increases complexity and decreases the explainability of predictions, which are important concerns in credit scoring. Further work might be concerned with explaining such predictions.

The GraphSAGE algorithm is inductive, which makes it usable over new samples or different time periods. Additionally, we have adjusted the amount features according to the inflation. However, the variance in the inflation-adjusted values shows that the model performance may degrade over time nonetheless. Therefore, both robust feature engineering and automatic machine learning adaptations might be future work directions as well.

BIBLIOGRAPHY

Aggarwal, C. C. (2018). *Neural Networks and Deep Learning*, volume 10. Springer.

Akoglu, L., Tong, H., and Koutra, D. (2015). Graph based anomaly detection and description: a survey. *Data mining and knowledge discovery*, 29:626–688.

Alpaydın, E. (2014). *Introduction to Machine Learning*. The MIT Press, Cambridge, Massachusetts.

Babaev, D., Savchenko, M., Tuzhilin, A., and Umerenkov, D. (2019). Et-rnn: Applying deep learning to credit loan applications. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 2183–2190.

Barnard, G. A. (1949). Statistical inference. *Journal of the Royal Statistical Society. Series B (Methodological)*, 11(2):115–149.

Basel Committee on Banking Supervision (2017). *Basel III: Finalising post-crisis reforms*. Bank for International Settlements (BIS).

Bengio, Y. (2012). Practical recommendations for gradient-based training of deep architectures. In *Neural networks: Tricks of the trade: Second edition*, pages 437–478. Springer.

Breiman, L. (2001). Random forests. *Machine learning*, 45:5–32.

Breiman, L. (2017). *Classification and regression trees*. Routledge.

Chen, T. and Guestrin, C. (2016). Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794.

Cheng, D., Tu, Y., Ma, Z., Niu, Z., and Zhang, L. (2019). Risk assessment for networked-guarantee loans using high-order graph attention representation. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*, pages 5822–5828. International Joint Conferences on Artificial Intelligence Organization.

CSIRO's Data61 (2018). Stellargraph machine learning library. `https://github.com/stellargraph/stellargraph`.

Dastile, X., Celik, T., and Potsane, M. (2020). Statistical and machine learning models in credit scoring: A systematic literature survey. *Applied Soft Computing*, 91:106263.

Draper, N. R. and Smith, H. (1998). *Applied regression analysis*, volume 326. John Wiley & Sons.

Fan, R.-E., Chang, K.-W., Hsieh, C.-J., Wang, X.-R., and Lin, C.-J. (2008). Liblinear: A library for large linear classification. *the Journal of machine Learning research*, 9:1871–1874.

Glorot, X. and Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. In Teh, Y. W. and Titterington, M., editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 249–256, Chia Laguna Resort, Sardinia, Italy. PMLR.

Gong, L. and Cheng, Q. (2019). Exploiting edge features for graph neural networks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 9211–9219.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. The Mit Press, Cambridge, Massachusetts.

Grover, A. and Leskovec, J. (2016). node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 855–864.

Guille, A., Hacid, H., Favre, C., and Zighed, D. A. (2013). Information diffusion in online social networks: A survey. *ACM Sigmod Record*, 42(2):17–28.

Hamilton, W., Ying, Z., and Leskovec, J. (2017). Inductive representation learning on large graphs. *Advances in neural information processing systems*, 30.

Hand, D. J. and Till, R. J. (2001). A simple generalisation of the area under the roc curve for multiple class classification problems. *Machine learning*, 45:171–186.

He, K., Zhang, X., Ren, S., and Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034.

Ho, T. K. (1995). Random decision forests. In *Proceedings of 3rd international conference on document analysis and recognition*, volume 1, pages 278–282. IEEE.

Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., Ye, Q., and Liu, T.-Y. (2017). Lightgbm: A highly efficient gradient boosting decision tree. *Advances in neural information processing systems*, 30.

Khazane, A., Rider, J., Serpe, M., Gogoglou, A., Hines, K., Bruss, C. B., and Serpe, R. (2019). Deeptrax: Embedding graphs of financial transactions. In *2019 18th IEEE International Conference On Machine Learning And Applications (ICMLA)*, pages 126–133. IEEE.

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Kipf, T. N. and Welling, M. (2016a). Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*.

Kipf, T. N. and Welling, M. (2016b). Variational graph auto-encoders. *arXiv preprint arXiv:1611.07308*.

Kubat, M. (2017). *An introduction to machine learning*. Springer.

LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *nature*, 521(7553):436–444.

Leskovec, J. (2023). Cs224w: Machine learning with graphs.

Li, Y., Tarlow, D., Brockschmidt, M., and Zemel, R. (2015). Gated graph sequence neural networks. *arXiv preprint arXiv:1511.05493*.

Li, Z., Wang, X., Yao, L., Chen, Y., Xu, G., and Lim, E.-P. (2022). Graph neural network with self-attention and multi-task learning for credit default risk prediction. In *International Conference on Web Information Systems Engineering*, pages 616–629. Springer.

Lillicrap, T. P., Santoro, A., Marris, L., Akerman, C. J., and Hinton, G. (2020). Backpropagation and the brain. *Nature Reviews Neuroscience*, 21(6):335–346.

Lundberg, S. M. and Lee, S.-I. (2017). A unified approach to interpreting model predictions. *Advances in neural information processing systems*, 30.

Mandrekar, J. N. (2010). Receiver operating characteristic curve in diagnostic test assessment. *Journal of Thoracic Oncology*, 5(9):1315–1316.

Mathew, S., Mordeson, J. N., and Binu, M. (2023). *Weighted and Fuzzy Graph Theory*. Springer Nature.

Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013). Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

Muñoz-Cancino, R., Bravo, C., Ríos, S. A., and Graña, M. (2023). On the combination of graph data for assessing thin-file borrowers' creditworthiness. *Expert Systems with Applications*, 213:118809.

Nwankpa, C., Ijomah, W., Gachagan, A., and Marshall, S. (2018). Activation functions: Comparison of trends in practice and research for deep learning. *arXiv preprint arXiv:1811.03378*.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

Perozzi, B., Al-Rfou, R., and Skiena, S. (2014). Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 701–710.

Schmidt, M., Le Roux, N., and Bach, F. (2017). Minimizing finite sums with the stochastic average gradient. *Mathematical Programming*, 162:83–112.

Shi, Y., Qu, Y., Chen, Z., Mi, Y., and Wang, Y. (2024). Improved credit risk prediction based on an integrated graph representation learning approach with graph transformation. *European Journal of Operational Research*, 315(2):786–801.

Shumovskaia, V., Fedyanin, K., Sukharev, I., Berestnev, D., and Panov, M. (2021). Linking bank clients using graph neural networks powered by rich transactional data. *International Journal of Data Science and Analytics*, 12:135–145.

Siddiqi, N. (2012). *Credit risk scorecards: developing and implementing intelligent credit scoring*, volume 3. John Wiley & Sons.

Siddiqi, N. (2017). *Intelligent credit scoring: Building and implementing better credit risk scorecards*. John Wiley & Sons.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958.

Sukharev, I., Shumovskaia, V., Fedyanin, K., Panov, M., and Berestnev, D. (2020). Ews-gcn: Edge weight-shared graph convolutional network for transactional banking data. In *2020 IEEE International Conference on Data Mining (ICDM)*, pages 1268–1273. IEEE.

Tang, J., Qu, M., Wang, M., Zhang, M., Yan, J., and Mei, Q. (2015). Line: Large-scale information network embedding. In *Proceedings of the 24th international conference on world wide web*, pages 1067–1077.

Tenenbaum, J. B., Silva, V. d., and Langford, J. C. (2000). A global geometric framework for nonlinear dimensionality reduction. *science*, 290(5500):2319–2323.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017). Attention is all you need. *Advances in neural information processing systems*, 30.

Velickovic, P., Cucurull, G., Casanova, A., Romero, A., Lio, P., Bengio, Y., et al. (2017). Graph attention networks. *stat*, 1050(20):10–48550.

Wang, D., Cui, P., and Zhu, W. (2016). Structural deep network embedding. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1225–1234.

West, D. B. (2002). *Introduction to graph theory*. Pearson, 2 edition.

Wu, F., Souza, A., Zhang, T., Fifty, C., Yu, T., and Weinberger, K. (2019). Simplifying graph convolutional networks. In *International conference on machine learning*, pages 6861–6871. PMLR.

Wu, J., Zhao, X., Yuan, H., and Si, Y.-W. (2023). Cdgat: a graph attention network method for credit card defaulters prediction. *Applied Intelligence*, 53(10):11538–11552.

Wu, L., Cui, P., Pei, J., and Zhao, L. (2022). *Graph Neural Networks: Foundations, Frontiers, and Applications*. Springer Nature Singapore, Singapore, 1st 2022. edition.

Xu, K., Hu, W., Leskovec, J., and Jegelka, S. (2018). How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826*.

Yang, C., Liu, Z., Zhao, D., Sun, M., and Chang, E. Y. (2015). Network representation learning with rich text information. In *IJCAI*, volume 2015, pages 2111–2117.

Yang, M., Meng, Z., and King, I. (2020). Featurenorm: L2 feature normalization for dynamic graph embedding. In *2020 IEEE International Conference on Data Mining (ICDM)*, pages 731–740. IEEE.

You, K., Long, M., Wang, J., and Jordan, M. I. (2019). How does learning rate decay help modern neural networks? *arXiv preprint arXiv:1908.01878*.

Zhang, M. and Chen, Y. (2018). Link prediction based on graph neural networks. *Advances in neural information processing systems*, 31.

Zhu, C., Byrd, R. H., Lu, P., and Nocedal, J. (1997). Algorithm 778: L-bfgs-b: Fortran subroutines for large-scale bound-constrained optimization. *ACM Transactions on mathematical software (TOMS)*, 23(4):550–560.

Appendix A

Table A.1: Target ratio, number of samples, first and second hop neighborhood NPL ratios of the test set by percentile differences of the new and the existing credit scoring model.

			NPL Ratio							
			First Hop Neighborhood				Second Hop Neighborhood			
Percentile Difference	Target Ratio	Number of Samples	Avg. Outgoing	Wght. Avg. Outgoing	Avg. Incoming	Wght. Avg. Incoming	Avg. Outgoing	Wght. Avg. Outgoing	Avg. Incoming	Wght. Avg. Incoming
[-65, -55)	0%	14	0.99%	0.26%	0%	0%	0.28%	0.46%	0.3%	0.59%
[-55, -45)	0%	126	0.45%	0.6%	0.74%	0.41%	0.57%	0.38%	0.33%	0.15%
[-45, -35)	0%	333	0.3%	0.21%	0.66%	0.84%	0.41%	0.43%	0.37%	0.23%
[-35, -25)	0.15%	652	0.65%	0.79%	0.44%	0.39%	0.53%	0.46%	0.43%	0.32%
[-25, -15)	0%	1027	0.58%	0.43%	0.42%	0.34%	0.39%	0.41%	0.36%	0.39%
[-15, -5)	0.37%	1603	0.43%	0.26%	0.44%	0.47%	0.56%	0.44%	0.44%	0.35%
[-5, 5)	2.6%	2541	0.57%	0.49%	0.44%	0.47%	0.51%	0.61%	0.44%	0.34%
[5, 15)	0.81%	1482	0.65%	0.61%	0.47%	0.49%	0.63%	0.64%	0.35%	0.3%
[15, 25)	0.21%	971	0.63%	0.69%	0.36%	0.33%	0.9%	0.87%	0.45%	0.36%
[25, 35)	0.19%	527	0.45%	0.36%	0.22%	0.13%	0.66%	0.72%	0.32%	0.21%
[35, 45)	0.33%	307	0.74%	0.82%	1.52%	1.36%	0.63%	0.48%	0.34%	0.35%
[45, 55)	0.55%	182	0.52%	0.9%	0.14%	0.03%	0.54%	0.39%	0.44%	0.68%
[55, 65)	1.02%	98	0.93%	1.27%	0%	0%	0.84%	1.81%	0.72%	0.49%
[65, 75)	3.12%	32	1.56%	2.97%	0%	0%	0.2%	0.04%	0.24%	0.08%
[75, 85)	0%	6	0%	0%	0%	0%	0.66%	1.08%	0.33%	0.23%

Table A.2: Target ratio, number of samples, first and second hop neighborhood existing credit model scores of the test set by percentile differences of the new and the existing credit scoring model.

			Existing Credit Score							
			First Hop Neighborhood				Second Hop Neighborhood			
Percentile Difference	Target Ratio	Number of Samples	Avg. Outgoing	Wght. Avg. Outgoing	Avg. Incoming	Wght. Avg. Incoming	Avg. Outgoing	Wght. Avg. Outgoing	Avg. Incoming	Wght. Avg. Incoming
[-65, -55)	0%	14	1.97	1.02	1.83	1.79	1.82	1.09	1.72	1.17
[-55, -45)	0%	126	1.69	1.48	1.83	1.83	2.16	1.5	2.02	1.82
[-45, -35)	0%	333	1.79	1.52	2.06	1.79	2.06	1.6	1.96	1.62
[-35, -25)	0.15%	652	1.77	1.56	1.99	1.74	2.13	1.55	2.04	1.62
[-25, -15)	0%	1027	1.76	1.62	2.03	1.87	2.17	1.79	2.11	1.82
[-15, -5)	0.37%	1603	2.07	1.94	2.22	2.09	2.35	2.08	2.26	1.95
[-5, 5)	2.6%	2541	2.46	2.53	2.69	2.57	2.82	2.64	2.86	2.53
[5, 15)	0.81%	1482	2.42	2.4	2.49	2.39	2.59	2.37	2.61	2.18
[15, 25)	0.21%	971	2.7	2.63	2.91	2.78	2.37	2.11	2.73	2.29
[25, 35)	0.19%	527	2.42	2.36	2.57	2.52	2.36	2.16	2.48	2.06
[35, 45)	0.33%	307	2.31	2.3	2.39	2.4	2.34	2.1	2.51	1.91
[45, 55)	0.55%	182	2.51	2.25	2.52	2.67	2.38	2.34	2.62	2.57
[55, 65)	1.02%	98	2.27	2.51	2.85	2.89	2.92	3.1	2.67	2.29
[65, 75)	3.12%	32	2.99	3.59	2.19	2.47	2.17	1.91	2.9	2.43
[75, 85)	0%	6	1.79	1.63	2.51	2.51	3.39	2.88	3.56	2.22