Factors Influencing Machine Learning Performance in Graph Analysis

Evangelos Iliadis

Diploma Thesis

Supervisor: Evaggelia Pitoura

Ioannina, September 2024

Τμημα Μηχ. Η/Υ & Πληροφορικησ
Πανεπιστημιο Ιωαννινων

Department of Computer Science & Engineering
University of Ioannina

Acknowledgements

This diploma thesis was completed as part of the undergraduate program of the Department of Computer Science and Engineering at the Faculty of Engineering, University of Ioannina.

I would like to express, my sincere gratitude to my supervisor, Professor Evaggelia Pitoura, for her invaluable collaboration and guidance. I would also like to extend my gratitude to Christos Gkartzios, PhD candidate, for his continuous assistance and invaluable support throughout the course of this research.

Ioannina, September 2024

Evangelos Iliadis

Abstract

In this thesis, we explore the influence of various factors on the performance of machine learning models, in the framework of Neo4j, a graph database management system. We examine the different embedding algorithms supported by Neo4j, such as Node2vec, FastRP, GraphSAGE and HashGNN. Furthermore, we observe the discrepancy in performance results across different machine learning models, when using identical embeddings. Finally, to gain insight into real-world datasets we present the Cypher-based queries we have developed and the results they yielded.

Keywords: machine learning, embedding algorithms, link prediction, graph databases, Neo4j, Cypher-based queries

Table of Contents

Chapter 1. Introduction

Graphs are structured representations of data. They consist of nodes and edges. These forms of structured data are used to represent entities (nodes) and the relationships (edges) between them. For example, in the context of a social network, nodes correspond to people, whereas edges are relationships between them, such as mutual friendships. We are focused on extracting meaningful information, related to the factors that contribute into the formation of these relationships between entities. Graph database management systems such as Neo4j, provide a framework that enables the efficient extraction of information. To analyze the intricate patterns of these graphs, we employed machine learning models. In our research, we explore the factors that can influence the machine learning graph analysis, leading to variations in results and consequently different interpretations of the real-world information. Machine learning models require training on the specific graph, we aim to analyze. It is necessary to transform the complex graph data into numerical representations that can be processed by the machine learning models. Therefore, in the framework of Neo4j, we utilize the embedding algorithms of Node2vec, FastRP, GraphSAGE, HashGNN to transform graph information into numerical vectors. These distinct algorithms employ different methodologies to capture graph information, which might lead to potential biases. Additionally, we examine different machine learning models and the results they produce when provided identical embeddings. Furthermore, we present the Cypher-based queries we have developed and we explain the information they provide. Finally, we provide experimental results highlighting the influence of these factors in the processes of machine learning and graph data analysis.

Chapter 2. Datasets and Data Preprocessing

2.1 Datasets

In this study, we analyzed five distinct, publicly available social network datasets. We tried to eliminate the bias introduced by the use of a single dataset, since model performance and therefore research conclusions might be affected by the specific characteristics of one individual dataset. To guarantee objectivity in our research findings, we selected datasets that cover a broad spectrum of varying graph characteristics (table 1). These graphs are heterogeneous with respect to number of nodes, number of edges, density, node average degree and number of features.

- **Ego-Facebook[1] [1]:** A network of users from the social media and social networking service Facebook. Nodes represent the users of Facebook. Edges are mutual friendships between users. Node features describe user profile information, with the interpretation being anonymized and obscured.
- **Feather-LastFM Asia[2] [2]:** A social network of users from Asian countries. Nodes represent users of the music streaming service LastFM. Edges are mutual friendships between users. Node labels denote the user nationality, while node features are artists liked by the user.
- **Musae-Facebook Large Page-Page[3] [3]:** A webgraph of verified Facebook pages. Nodes represent pages. Edges are mutual likes between pages. Node labels describe page types, while node features denote page description information.

[1] https://snap.stanford.edu/data/ego-Facebook.html

[2] https://snap.stanford.edu/data/feather-lastfm-social.html

[3] https://snap.stanford.edu/data/facebook-large-page-page-network.html

- **Gemsec-Deezer (Croatia)[4] [4]:** A social network of users from the country of Croatia. Nodes represent users of the music streaming service Deezer. Edges are mutual friendships. Node features denote liked genre lists.
- **Gemsec-Deezer (Hungary)[4] [4]:** A social network of users from the country of Hungary. Nodes represent users of the music streaming service Deezer. Edges are mutual friendships. Node features denote liked genre lists.

Table 1: Statistics of datasets used.

Dataset	Nodes	Edges	Average Degree	Train Graph Edges	Test Graph Edges
Ego-Facebook (Largest WCC)	3927	84210	42.888	75789	8421
Feather-LastFM	7624	27806	7.294	25026	2780
Musae-Facebook	22470	171002	15.22	153902	17100
Gemsec-Deezer (Croatia)	54573	498202	18.258	448382	49820
Gemsec-Deezer (Hungary)	47538	222887	9.377	200599	22288

2.1.1 Dataset Features

For each dataset, we examined attributes available across all graph nodes. These are the features that we examined for each dataset case:

- **Ego-Facebook:** This social network case contains anonymized user profile information. In the context of this specific dataset case, we focused on the node features of gender, education concentration, education type and language.
- **Feather-LastFM Asia:** In the context of the LastFM social network we focused on the list of liked artists for each individual user. Furthermore, we included the node labels representing user nationality, that are provided for the classification task, as node features.
- **Musae-Facebook Large Page-Page:** In this webgraph case, we focused on the list of anonymized information regarding each verified Facebook page. Additionally, we included the node labels representing page categories, that are provided for the classification task, as node features.

[4] https://snap.stanford.edu/data/gemsec-Deezer.html

- **Gemsec-Deezer (Croatia, Hungary):** In the context of the Deezer dataset cases, we focused on the list of genre preferences for each individual user.

2.2 Data Preprocessing

Ensuring that the dataset graphs are connected is a crucial part for the process of embedding generation. The embedding algorithms utilize graph structure and node features to accurately represent graph information. A disconnected graph isolates certain nodes or entire graph components, distorting the graph patterns utilized by the model. We approached cases of such disconnected graphs (e.g. ego-Facebook dataset), by analyzing the largest weakly connected components (WCC), which is the subgraph that contains the highest number of connected nodes.

In the phase of data preprocessing, we ensured that no duplicate edges exist in the graph. The terms duplicate, parallel edges denote the presence of multiple edge instances, between the same pair of nodes. We removed all duplicate edges, in cases where multiple instances were detected, preserving only one, between the same pair of nodes. Embedding algorithms utilizing mechanisms such as node sampling through random walks and node feature aggregation, might be negatively influenced regarding the quality of the generated embeddings. Consequently, the existence of such edges might artificially increase the representation and centrality of certain nodes or features, by limiting the pattern diversity captured by the embeddings.

In the context of queries, node features are obligatory. We specifically selected datasets that provide node features and labels, including these multiclass node labels as node attributes. The introduction of features to the graph can significantly impact the experimental evaluation findings. In dataset cases (e.g. ego-Facebook) where distinct nodes had varying features, we isolated and retained only those available across all graph nodes. Since embedding generating algorithms prohibit incorporating node features that are not present in all graph nodes, to effectively mitigate bias introduced by inconsistent data. Embedding algorithms utilize numerical features in their inherent computational procedures. Specifically, HashGNN relies on features of binary representation. Thus, we transformed feature cases, such as multi-class node labels, into binary representation equivalent ones.

In order to guarantee the integrity of the model training methodology, we split the ground truth graph into training and test graphs. We ensured that the training graph connectivity remained intact after the split, with no isolated nodes or graph components. The test graph consists of a set of ground truth edges. In our case, we isolated 10% of the graph edges, from each individual dataset. We selected datasets that provide graphs with node average degrees that allow the extraction of edges. Dividing the original graph into two separates prevents any potential data leakage. Thus, the research findings produced will be entirely based on the model's generalization capability.

Chapter 3. Embeddings

Graph node embeddings play a crucial role in the field of data science and machine learning. The term embedding refers to numerical vectors that transform complex, high-dimensional, graph data into vectors of lower dimensions, while maintaining most of the important graph properties, such as graph structure, node and edge properties (Figure 1). A higher vector dimension provides more information-dense embeddings, while increasing storage and processing complexity.

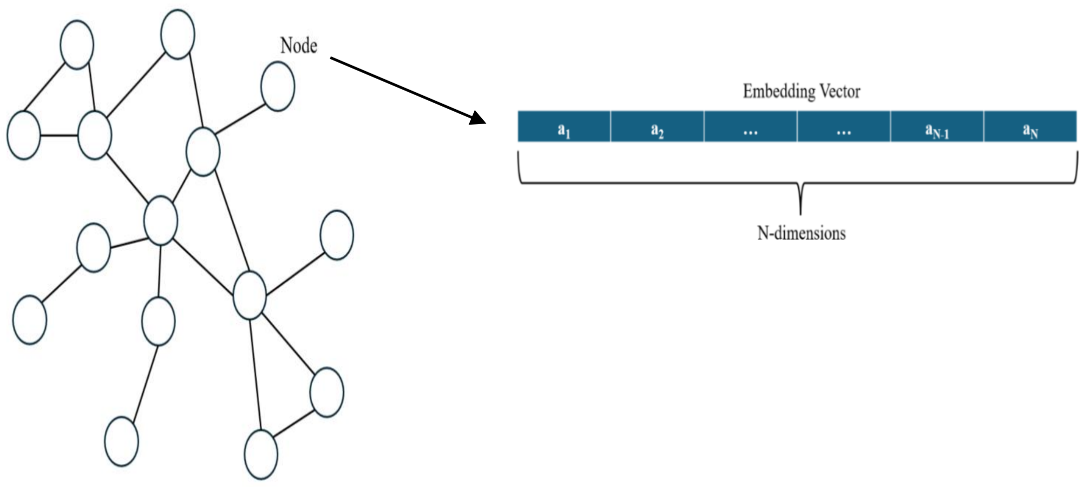
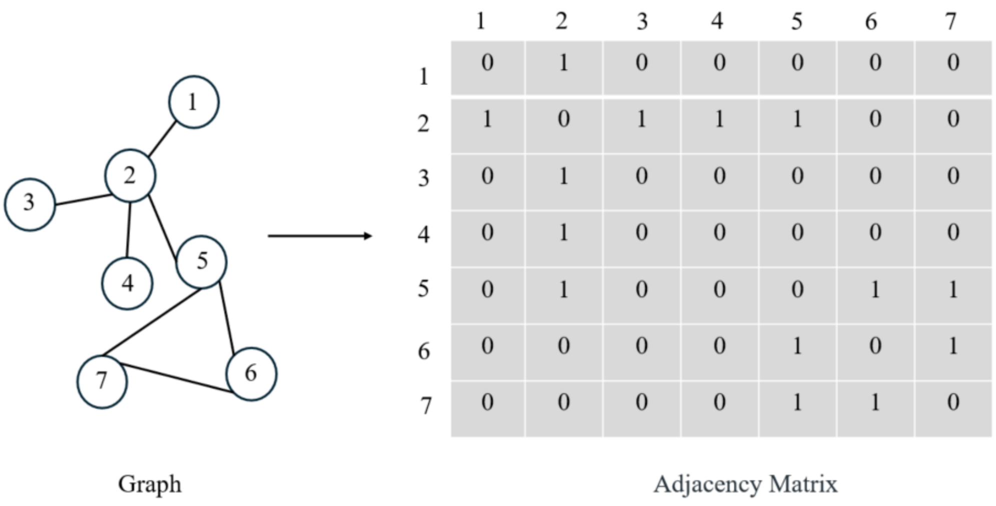


Figure 1: Visual representation of the node information being captured by the embedding vector.

An alternative approach to the use of embeddings includes the creation of N x N adjacency matrices, where N is the number of nodes in the graph (Figure 2). The complexity O(N^2) of an adjacency matrix scales quadratically, as the graph grows in size, demanding more resources. Similarly, to the representation of graph structure, node and edge features would require distinct adjacency matrices, where rows, columns correspond to the nodes/ edges and features respectively.

Thus, the encoded embedding information offers a unified solution of both structural and feature information representation. In this thesis, we utilize 128-dimensional vectors, that offer a balance between capturing essential graph information and computational efficiency.



	1	2	3	4	5	6	7
1	0	1	0	0	0	0	0
2	1	0	1	1	1	0	0
3	0	1	0	0	0	0	0
4	0	1	0	0	0	0	0
5	0	1	0	0	0	1	1
6	0	0	0	0	1	0	1
7	0	0	0	0	1	1	0

Figure 2: Visual representation of graph information capturing, using an adjacency matrix. The matrix elements indicate the presence (1), absence (0) of node links.

3.1 Graph Projection

Prior to utilizing the embedding generating algorithms in Neo4j, we must create a projection of the graph. This step creates a representation of the sub-graph, on which we want to perform our analysis. This step is useful in avoiding noise and data leakage. In our case, we eliminated these factors by splitting the original graph into two separate train and test graphs in the data pre-processing phase. Therefore, in this phase we projected the entirety of the train graph.

3.2 Embedding Generating Algorithms

In this section, we present the distinct embedding algorithms we incorporated to conduct our research, that are supported by the latest release of Neo4j (version 5.22). These are Node2Vec, Fast Random Projection (FastRP), GraphSAGE (Graph SAmple and aggreGatE) and HashGNN. Each one provides unique benefits in the process of embedding creation, depending on their respective mechanism of action.

3.2.1 Node2vec

Node2vec [5] relies on the graph structure, while disregarding the node properties for embedding generation. It captures the graph structure information using neighborhood sampling, by deploying biased random walks throughout the graph. The random walk starts at the corresponding graph node, whose embedding vector is being generated. Node2vec balances between the graph traversal algorithms of Breadth-first Sampling (BFS) and Depth-first Sampling (DFS). It is able to emulate the traversal behavior of BFS and DFS, by adjusting the parameters that influence the random walks. Specifically, the bias α is responsible for the sequence of nodes visited by the random walk. The adjustment of α is achieved by defining the parameters p and q. The return parameter, p influences the probability of revisiting a node during the random walk. A high value ($> \max(q,1)$) ensures that we are more likely to sample an unvisited node in the following two hops. The in-out parameter, q influences the exploration towards "inward" or "outward" nodes. If $q > 1$, the random walk is biased towards nodes close to the t node (Figure 3). Setting both $p = 1$ and $q = 1$ simulates a uniform random walk. This is a Node2vec sampling strategy sub-case called DeepWalk, where $p = q = 1$. The bias α of the random walk is equal to:

$$\alpha_{pq}(t, x) = \begin{cases} 1/p & \text{if } d_{tx} = 0 \\ 1 & \text{if } d_{tx} = 1 \\ 1/q & \text{if } d_{tx} = 2 \end{cases}$$

where d_{tx} represents the shortest path between nodes t and x.

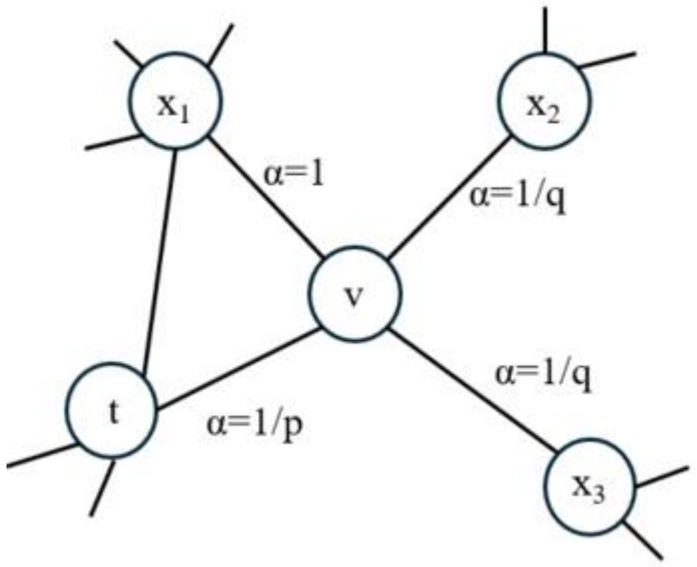


Figure 3: Visual representation of the Node2vec random walk procedure. The random walk just reached node v, after sampling node t. The bias α influences the likelihood of the next step for each neighboring node.

During the embedding generation phase, we define the walk length and walks per node. The term walk length represents the number of nodes visited during each random walk sequence. A higher value influences the walk into capturing extensive structural information from the graph. Extending the walk length helps unravel complex patterns that may not be apparent with shorter random walks. Walks per node refer to the number of random walks executed for each starting node. Increasing the number of walks per node influences the embeddings to capture more complex structural patterns. Higher values provide more graph information, at the expense of computational complexity. There is a threshold over which increasing the parameters of walk length and walks per node influence negatively the performance of the trained model, introducing more noise and overfitting.

3.2.2 Fast Random Projection (FastRP)

FastRP [6] utilizes random projection matrices. It relies on the mathematical theory of Johnson-Lindenstrauss Lemma. The Lemma states that it is possible to reduce the dimensionality of a set of node data points while preserving the distances between them. FastRP achieves dimensionality reduction by multiplying the n x m graph feature matrix M, by an m x d random projection matrix R. The product is an n x d matrix N, where d<<m. In graph data cases, n = m = number of nodes in the graph. FastRP utilizes node properties in the embedding generation process. The parameter propertyRatio controls the node property portion of the embedding vector. The embedding vector consists of the concatenation of two distinct parts (Figure 4). The first one captures graph structure information, while the second one captures property value information of neighboring nodes:

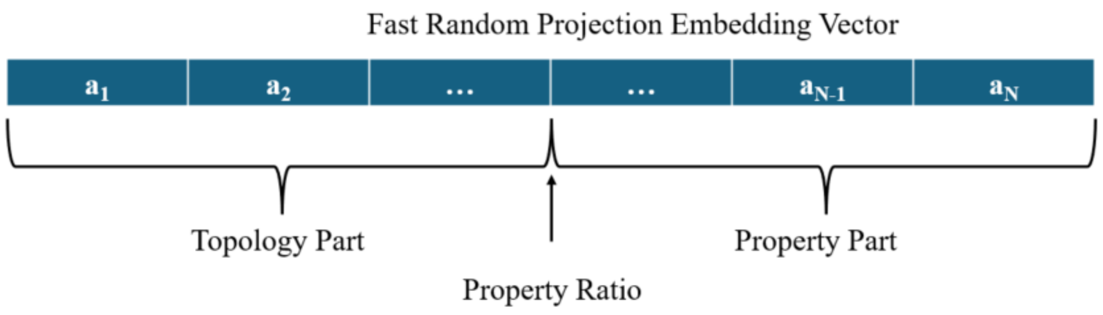


Figure 4: Visual illustration of a N-dimensional embedding vector, utilizing the FastRP algorithm.

The algorithm supports the extreme case scenarios, when propertyRatio = 0.0 or propertyRatio = 1.0 respectively. A value of propertyRatio = 0.0 influences the algorithm to base embedding generation entirely on the topological structure of the graph, eliminating the node feature part from the vector. Setting propertyRatio = 0.5 provides a balanced approach towards capturing both types of graph information into embedding vectors.

3.2.3 GraphSAGE (Graph SAmple and aggreGatE)

GraphSAGE [7] leverages node features to learn embedding aggregator functions that generalize to unseen node inductively, instead of transductively generating individual embedding vectors for each graph node. These functions aggregate feature information from the node neighborhood and demand the presence of node features to operate. It is extremely useful in dynamic graphs where new nodes are added, without the need for retraining. However, for the purposes of link prediction in our research, we are not focusing on dynamic graphs. The algorithm samples adjustable sets of neighborhood node layers and for each individual node, it generates embeddings by recursively propagating the node information accumulated from multiple layers of neighboring nodes (Figure 5). There are three main aggregation methods (Mean, Pooling, LSTM) but only two of them (Mean, Pooling) are supported by the current GraphSAGE implementation of the latest release of Neo4j.

- Mean Aggregator: Calculates the mean of the neighboring nodes feature vectors:

$$h_v^k \leftarrow \sigma(W * MEAN(\{h_v^{k-1}\} U \{h_u^{k-1}, \forall\, u \ \in N(v)\})$$

Breakdown of the Mean Aggregation function formula:
  - h_v^k : The embedding of node v in the k layer.
  - h_v^{k-1} : The embedding of node v in the previous layer k-1.
  - $h_u^{k-1}, \forall\, u \ \in N(v)$: The embedding of a neighboring node u, where u ∈ N(v) is the neighborhood of the node v in the previous layer k-1.
  - MEAN: The mean function that aggregates the embeddings from the previous layer of node v and its neighbors u.
  - W: A learnable weight matrix.
  - σ: The activation function e.g. ReLU, sigmoid, where ReLU is defined as ReLU(x) = max(0,x) and the logistic sigmoid as $\sigma(x) = \frac{1}{(1+ e^{-x})}$.

The mean aggregator is useful when the average of the node neighborhood is significant. Such cases are social network graphs.

- Pooling Aggregator: Applies a max-pooling operation to aggregate features from neighboring nodes.

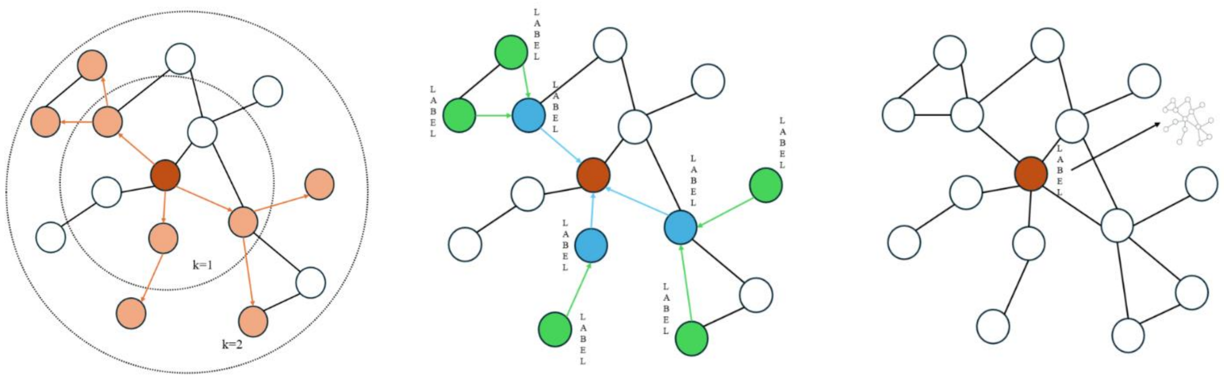
$$AGGREGATE_k^{pool} \leftarrow \max\,(\{\sigma\big(W_{pool} h_{u_i}^k + b\big), \forall u_i \in N(v)\})$$

Breakdown of the Pool Aggregation function formula:
  - $AGGREGATE_k^{pool}$: The embedding of node v in the k layer.

- $h_{u_i}^k$: The embedding of a neighboring node u_i in the layer k, where $u_i \in N(v)$ is the neighborhood of the node v.
- W_{pool} : A learnable weight transformation matrix.
- b_{pool} : A bias term added to the transformation of the neighbor embeddings.
- σ: The activation function e.g. ReLU, sigmoid.
- max : The max-pooling operation that is applied. It computes the maximum value of each feature across all the neighbors of node v.

It is useful when there are significant neighborhood anomalies. Such cases include scientific datasets e.g. chemistry.

- LSTM Aggregator: Uses a Long Short-Term Memory (LSTM) network to aggregate neighborhood features in a sequential manner.



From left to right: 1) Sample Neighborhood, 2) Aggregate feature information from neighboring nodes, 3) Predict graph context using aggregated information.

Figure 5: Visual illustration of the GraphSAGE approach. K represents the node layer level. Each individual node label is influenced by the aggregated labels of its neighbors.

Generating GraphSAGE embeddings in Neo4j, utilizes the procedure of node feature aggregation, which is achieved through model training. There are several parameters impacting the training of the model, such as activation function, number of layers, sample sizes, learning rate and epochs. The activation function affects the way the model processes the aggregated node features at each layer. Both activation functions (ReLU, logistic sigmoid) are non-linear, allowing the GraphSAGE model to learn more complex patterns. Applying ReLU (Rectified Linear Unit) to the aggregated features sets negative values to 0, while preserving the positive ones. The selection of the activation function is very important, because a poor choice can potentially hinder the model learning and ultimately the graph information provided by the embeddings.

The number of node layers is crucial in the process of node feature aggregation. A low value of layers captures information from the immediate node neighborhood, while increasing the number leads capturing wider graph patterns. Although increasing the layer number provides more aggregated feature information on a global graph scale, it is accompanied by the expense of high processing times and storage resource consumption. Furthermore, multiple layers introduce the risk of oversmoothing or overfitting. The term oversmoothing refers to the phenomenon where distinct nodes have almost identical embedding vectors. This is the result of the feature aggregation from distant nodes that hinder the recognition of neighboring nodes and their individual feature characteristics. Moreover, the size of the node sample influences the amount of neighboring node features aggregated. Increasing the size of the sampled nodes leads to capturing more complex feature information of the node neighborhood. The term learning rate denotes the step of the model towards convergence in the optimization process of the loss function. In cases of optimization tasks, a higher step decreases the time towards convergence but increases the risk of missing the optimal solution. On the other hand, a very small step size poses the risk of optimization process stopping at a local minimum, mistakenly recognizing it as the optimal solution. Lastly, epochs refer to the number of times the same data set is passed through the model. While higher numbers provide better performance, they also increase the computational complexity and provide diminishing returns after a specific threshold.

3.2.4 HashGNN

The HashGNN [8, 9] method uses graph neural networks to generate node embeddings, capturing both structural and node feature information in the process. Instead of high-dimensional embeddings, it generates binary hash codes, achieving storage complexity reduction, while guaranteeing graph structural information preservation. The graph neural network learns node embedding representations, using neighborhood node feature aggregation. Those embeddings are passed through the hash layer to get converted to binary hash codes, utilizing tanh, sign activation functions and the Bernoulli distribution. The hyperbolic tangent tanh and sign activation functions are defined as:

$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$ and $sign(x) = \begin{cases} 1\ if\ x > 0 \\ 0\ if\ x = 0 \\ -1\ if\ x < 0 \end{cases}$ respectively. The formula for the probability mass function of the discrete Bernoulli distribution is:

$$P(X = k) = p^k(1-p)^{1-k}$$

Breakdown of the probability mass function formula for the Bernoulli distribution:

- X is the random variable representing the outcome.
- k is the outcome, where k ∈ {0,1}. k = 1 denotes the existence of an edge, while k = 0 represents the absence of an edge.
- p is the probability of success, i.e., the presence of an edge, where X = 1.
- 1-p is the probability of failure, i.e., the absence of an edge, where X = 0.

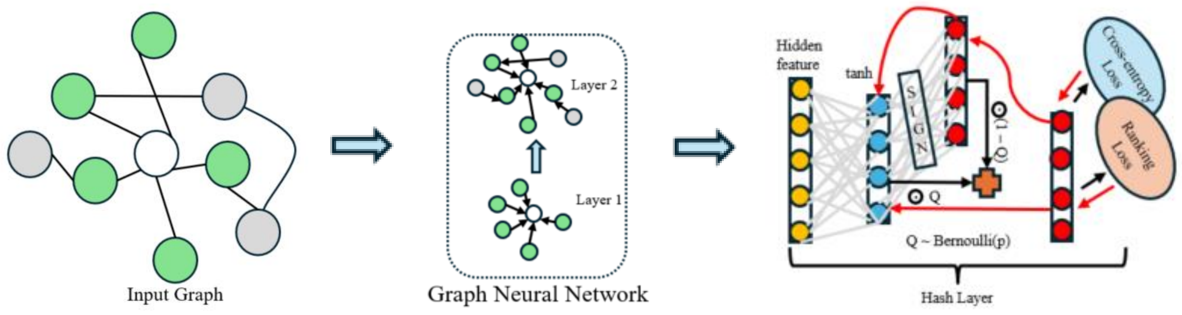


Figure 6: Architecture of the HashGNN.

This algorithm supports embedding generation based on two distinct methodologies. One relies on the graph topological structure, while the other incorporates both graph structure and node feature aggregation. Notably, HashGNN cannot operate in the extreme case where the graph structure is excluded and the embedding vectors are entirely based on node feature aggregation. Similarly, to the generated binary hash codes, the HashGNN algorithm requires features in binary representation to function. Any non-binary feature should be converted accordingly. The implementation of the HashGNN architecture in Neo4j and therefore the information of the embeddings is influenced by the parameter, embeddingDensity. This parameter controls the sparsity of the embedding vector, with higher values decreasing the number of zeros included in the binary hash code. An embedding vector populated by more non-zero elements, expresses more information regarding the graph structural patterns and its features.

In summary, we have explored the Neo4j implementations for the embedding generating algorithms of Node2vec, FastRP, GraphSAGE and HashGNN. Each one approaches the process of capturing graph information uniquely, providing distinct embedding vectors for identical graphs. Therefore, we expect variations in our findings produced during the phase of link prediction.

Chapter 4. Link Prediction

In section 3, we generated embeddings, using various embedding generating algorithms, that are fundamental for the link prediction process. The term link prediction refers to the forecast of future or missing relationships, based on the structural and feature pattern information captured by the embedding vectors. On a graph level, link prediction involves the addition of new edges amongst already existing nodes. The task of link prediction can be framed as a classification problem. This problem involves 2 classes, with class 1 representing the positive edges (existing or future predicted ones) and class 0 representing the negative edges (non-existent that do not belong to the graph). The generated embeddings will be used along with sets of positive and negative edge examples to train the model responsible for the link prediction, during the model learning process. The training and testing edges are labeled accordingly to indicate whether an edge instance is positive or negative. This way, the model is trained on sets of examples where it is given the correct classification answers. The goal is to train the model to develop optimal generalization capability, i.e., the ability to distinguish between the two classes for unknown, non-provided during the learning process, data. This approach, where the model learns from labeled examples, is referred to as supervised learning.

4.1 Example Set Creation

Examples are sets of positive and negative edge instances, utilized in the processes of training and testing the model. Consequently, we created sets of positive train examples, negative train examples, positive test examples and negative test examples. The positive example sets consist of the entirety of the train or test graph edges respectively. While the negative example sets consist of a sub-set of the non-existent train graph or test graph node pairs respectively. Since the actual amount of the non-existent edges is larger than the amount of the existing ones, only a portion of randomly sampled edges, equal to the amount of the existing ones, is kept for the two negative sets. Notably, we ensured that there were no overlaps among the sets, to prevent data leakage from altering our conclusions. This way, an equal distribution of both class examples, is used to learn and evaluate the presence or absence of edges thoroughly. During the learning process the model identifies common characteristics and patterns among true positive and true negative instances. The example sets that we use in each individual dataset are the same for all embedding types, to ensure fairness and consistency to all types, in the evaluation process.

4.2 Training and Testing Data Creation

A classification model bases its decisions on embeddings. To create the training (X_train, Y_train) and testing (X_test, Y_test) datasets, for edge prediction, we transition from node embeddings to edge embeddings. One approach involves concatenating the two 128-dimensional embedding vectors into a 256-dimensional vector. However, we observed that this approach of creating edge embeddings lowered significantly the performance of the model. Instead, we opted for the Hadamard product notation, using the binary operator $\boxdot$. This operation performs element-wise multiplication of the corresponding node embedding elements, combining them into an edge embedding element:

$$[f(u) \boxdot \mathrm{f(v)}]_i = f_i(u) * f_i(v)$$

Breakdown of the Hadamard product notation formula:

- u, v = The nodes at the endpoints of the edge.
- f(u), f(v) = The embedding vectors of nodes u, v.
- $f_i(u)$, $f_i(v)$ = The i-th component of the vectors f(u), f(v) accordingly.
- $[f(u) \boxdot f(v)]_i$ = The i-th component of the product edge embedding vector.

This approach resulted in compact, information-dense embeddings that provide better model performance, while reducing computational complexity.

4.3 Evaluation Metrics

To effectively compare the influence of different embedding types and models on our research findings, it is essential to understand the evaluation metrics. In our case, the metrics show insights into the potential of the model in predicting links accurately. Prior to understanding the evaluation metrics, let us look at the following confusion matrix (Figure 5):

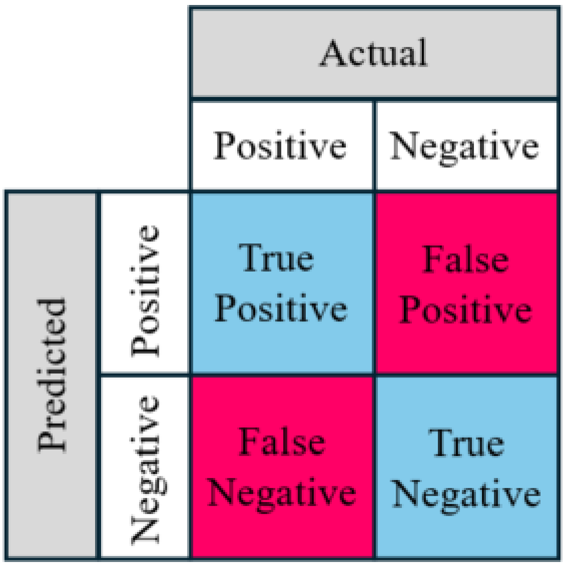


Figure 7: Confusion Matrix for a Classification Task

- True Positive: Actual positive edges that the model correctly predicted as positive
- True Negative: Actual negative edges that the model correctly predicted as negative
- False Positive: Actual negative edges that the model falsely predicted as positive
- False Negative: Actual positive edges that the model falsely predicted as negative.

There are several metrics used to showcase model performance, each one providing different perspectives into how well the model predicts different classes and classification scenarios. In this phase, we introduce the evaluation metrics of Accuracy, Precision, Recall (Sensitivity/ True Positive Rate), F1 Score and ROC AUC (Receiver Operating Characteristics Area Under Curve).

- Accuracy: The percentage of correct predictions. It is the ratio of correctly predicted edges (positive and negative), divided by the total amount of predicted edges. In our case, where both classes are represented with equal sets of positive and negative examples, it is a reliable indicator of the model's predicting capability. The formula for the calculation of the Accuracy metric is defined as follows:

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

- Precision: The ratio of correctly predicted positive edges, divided by the number of edges predicted as positive. A higher precision value ensures an increased probability that a predicted positive instance is correct. The formula for the calculation of the Precision metric is defined as follows:

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

- Recall/ Sensitivity/ True Positive Rate (TPR): The representation of the ability of the model to correctly identify positive instances. A high value indicates that the model correctly identifies most of the actual positive edges. The formula for the calculation of the Recall metric is defined as follows:

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

- F1 Score: The harmonic mean of Precision and Recall. The formula for the calculation of the F1 Score metric is defined as follows:

$$F1\ Score = 2\ x\ \frac{Precision\ x\ Recall}{Precision + Recall}$$

- ROC AUC (Receiver Operating Characteristics Area Under Curve): It represents the area under the ROC curve. It is a plot of the TPR (Recall/ Sensitivity) on the y-axis, against the False Positive Rate (FPR) on the x-axis. False Positive Rate is defined as:

$$FPR = \frac{FP}{TN + FP}$$

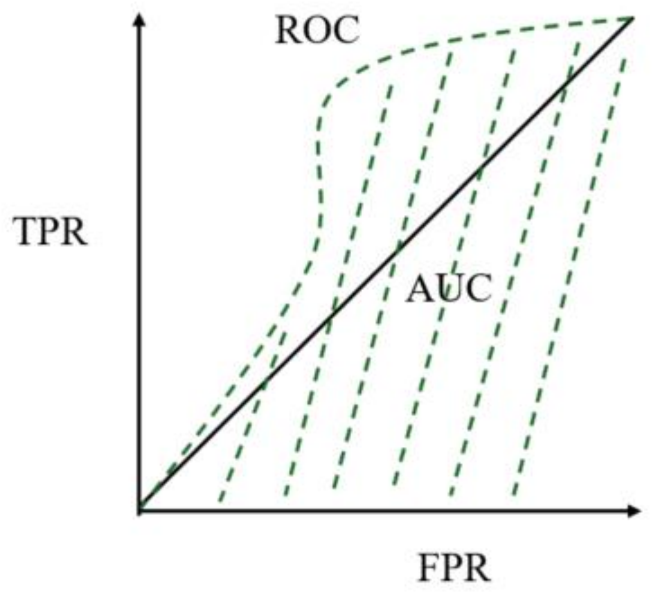


Figure 8: Visual illustration of the ROC AUC classification evaluation metric.

4.4 Model Selection and Training

In the field of machine learning, models are fundamental components. They are used for characteristic pattern recognition in unsupervised learning and predictive assessment in supervised classification problems involving labeled datasets. Selecting the appropriate model type is crucial, as it can drastically impact the research findings. Since the problem of link prediction can be framed as a classification problem between two classes we employed the Logistic Regression model (Figure 9), because it combines reliable performance in binary classification tasks and low storage and computational complexity. The logistic regression model calculates the probability of the binary class outcomes, of the input dataset. The model utilizes the logistic activation function σ(x) = $\frac{1}{1+e^{-x}}$,which provides a non-linear sigmoid curve, that allows for the approach to more complex classification problems, than the linear regression hyperplane.

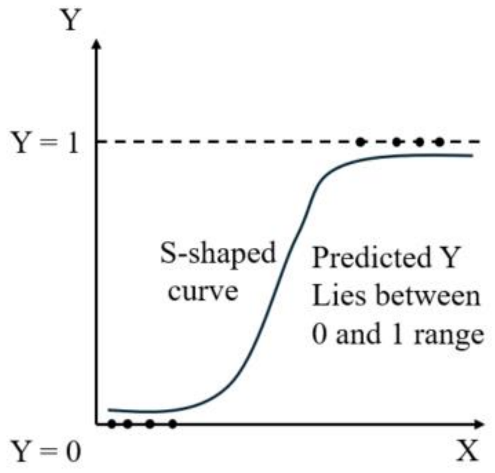
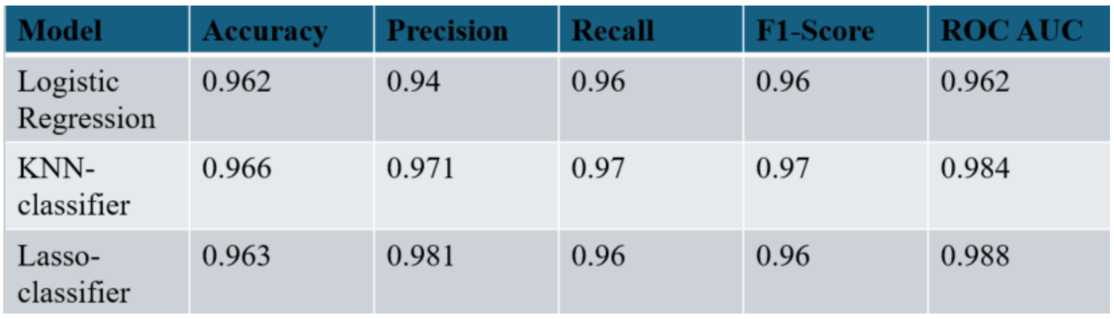


Figure 9: Visual illustration of the sigmoid curve of the logistic regression model. The model calculates the probability of the input data. The S-shaped curve separates the decisions made for each class.

We derived from our research that models such as K-Nearest Neighbors (KNN) classifier and Lasso (Least Absolute Shrinkage and Selection Operator) classifier provide slightly improved evaluation metrics than logistic regression. However, the KNN and Lasso models have increased computational complexity, with longer execution times, especially in the large datasets. KNN stores the entire dataset, to calculate the distances (Euclidean, Manhattan, cosine similarity) between each data point pair, whereas Lasso involves L1 regularization process. Considering the factors of predictive accuracy and resource consumption, the logistic regression model provides the optimal choice, achieving similar performance to the others while minimizing computational overhead.

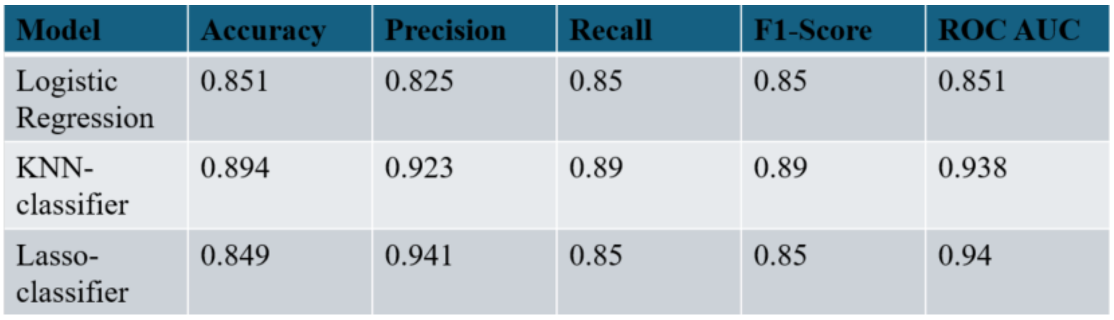
In this section, we present the comparison of performance evaluation metrics for the best performing classification models, applied to different datasets. To guarantee the consistency and integrity of our findings, we conducted our experiment, utilizing identical Node2vec embeddings across the various models. By preserving all other factors identical, we were able to eliminate any potential fluctuations introduced by the use of different Node2vec embeddings and isolate the impact of each individual model on the performance evaluation metrics. Here we present the matrices, containing detailed model performance information for the datasets of ego-Facebook, feather-LastFM, musae-Facebook, gemsec-Deezer (Croatia) and gemsec-Deezer (Hungary):

Table 2: Performance evaluation metrics of different models utilizing Node2vec embeddings, regarding the ego-Facebook dataset.



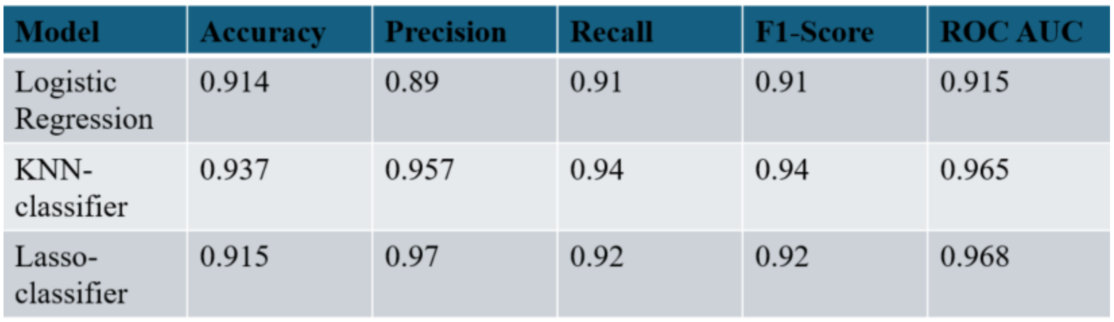
Model	Accuracy	Precision	Recall	F1-Score	ROC AUC
Logistic Regression	0.962	0.94	0.96	0.96	0.962
KNN-classifier	0.966	0.971	0.97	0.97	0.984
Lasso-classifier	0.963	0.981	0.96	0.96	0.988

Table 3: Performance evaluation metrics of different models utilizing Node2vec embeddings, regarding the feather-LastFM dataset.



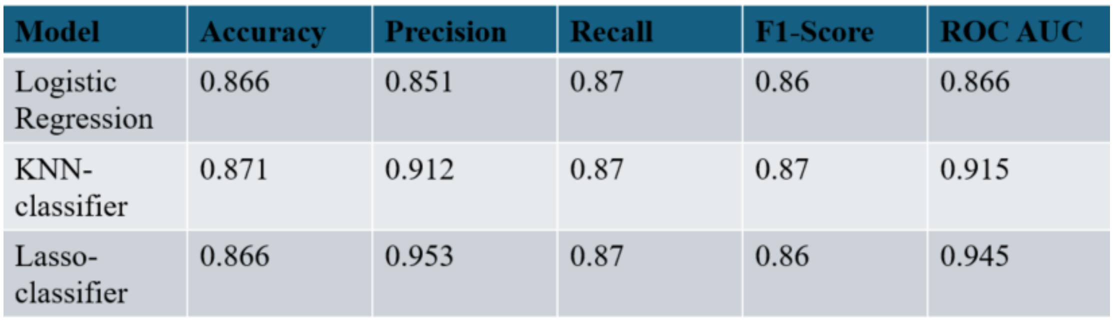
Model	Accuracy	Precision	Recall	F1-Score	ROC AUC
Logistic Regression	0.851	0.825	0.85	0.85	0.851
KNN-classifier	0.894	0.923	0.89	0.89	0.938
Lasso-classifier	0.849	0.941	0.85	0.85	0.94

Table 4: Performance evaluation metrics of different models utilizing Node2vec embeddings, regarding the musae-Facebook dataset.



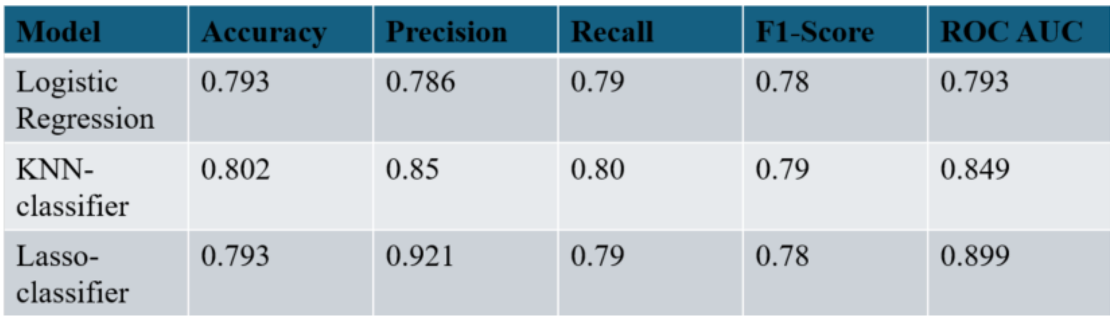
Model	Accuracy	Precision	Recall	F1-Score	ROC AUC
Logistic Regression	0.914	0.89	0.91	0.91	0.915
KNN-classifier	0.937	0.957	0.94	0.94	0.965
Lasso-classifier	0.915	0.97	0.92	0.92	0.968

Table 5: Performance evaluation metrics of different models utilizing Node2vec embeddings, regarding the gemsec-Deezer (Croatia) dataset.



Model	Accuracy	Precision	Recall	F1-Score	ROC AUC
Logistic Regression	0.866	0.851	0.87	0.86	0.866
KNN-classifier	0.871	0.912	0.87	0.87	0.915
Lasso-classifier	0.866	0.953	0.87	0.86	0.945

Table 6: Performance evaluation metrics of different models utilizing Node2vec embeddings, regarding the gemsec-Deezer (Hungary) dataset.



Model	Accuracy	Precision	Recall	F1-Score	ROC AUC
Logistic Regression	0.793	0.786	0.79	0.78	0.793
KNN-classifier	0.802	0.85	0.80	0.79	0.849
Lasso-classifier	0.793	0.921	0.79	0.78	0.899

Notably, the KNN classification model consistently outperformed the others in terms of accuracy, Recall and F1-Score, while the Lasso classification model yielded the highest precision and ROC AUC, in all dataset cases. Although, the KNN and Lasso classifiers provide slightly better performance than logistic regression, the differences in metrics are marginal, occurring in the second or third decimal place. This supports our decision to prioritize a balance of process complexity and quality in our findings.

Having established that the Logistic Regression model is suitable for our research purposes, we proceed to the model fitting. In this step, the selected classifier model is trained on the dataset, prepared in section 4.2 of our thesis. A model that is optimally trained has its weights and biases adjusted, in such a way to identify the specific dataset examples correctly. We stated that the Logistic Regression model utilizes the logistic sigmoid activation function:

$$\sigma(u) = \frac{1}{1+e^{-u}}, \textit{ where } u = w^T x + b$$

- W: The learnable weights.
- x: The input data vector.
- b: A bias constant that is included in the calculation of u.

The objective of the learning process is to minimize the loss function over the specific dataset being presented, using gradient descent to iteratively adjust the model weight and bias parameters. The process relies on the training datasets X_train and Y_train. The X_train set consists of positive and negative edge embedding vectors, whereas Y_train contains the actual labels corresponding to the edges, with 1 indicating a positive instance and 0 a negative one.

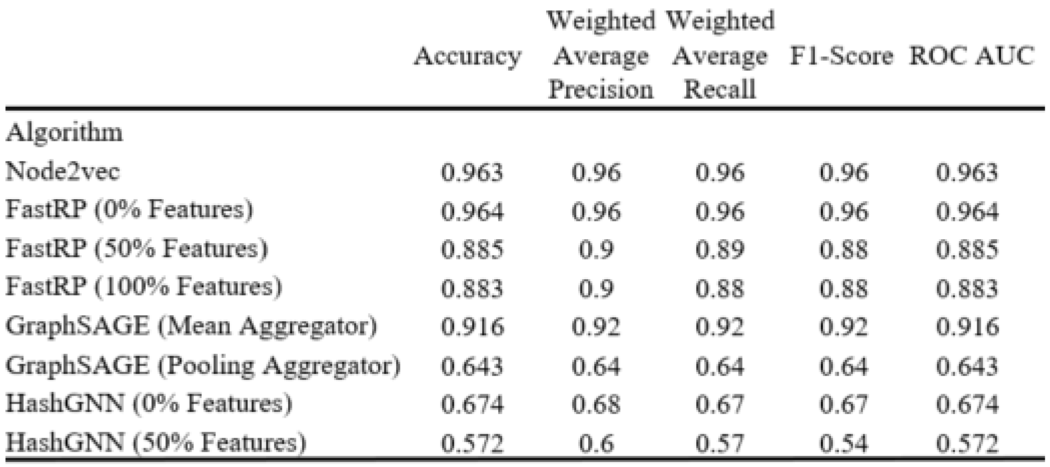
4.5 Embedding Performance Evaluation

In the previous stage, we trained the logistic regression model. Before we begin with the prediction of links, we need to evaluate its performance. In this section we provide unseen during the training process data(X_test), to assess the ability of the model to generalize.

Here we present matrices containing detailed performance metric information for the datasets of ego-Facebook, feather-LastFM, musae-Facebook, gemsec-Deezer (Croatia) and gemsec-Deezer (Hungary), using different embedding types. The first matrix, for each dataset, illustrates the logistic regression model performance to recognize examples regarding both classes 1 and 0, for various embedding types. The second matrix, for each dataset, focuses on the ability of the model to identify the true positives of the ground truth. Lastly, the heatmaps highlight overlaps between different embedding types, regarding predicted true positives. Notably, we ensured that all mutual parameters and conditions were identical across the different embedding types to isolate the impact each embedding might have on model performance. Here we present variations in model performance utilizing the distinct embedding types:

Ego-Facebook Dataset

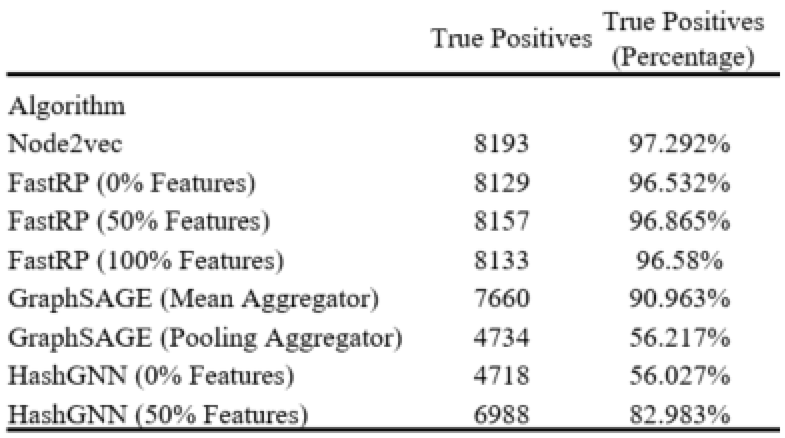
Table 7: Logistic Regression performance evaluation metrics for different embedding types, regarding the ego-Facebook dataset.



	Accuracy	Weighted Average Precision	Weighted Average Recall	F1-Score	ROC AUC
Algorithm					
Node2vec	0.963	0.96	0.96	0.96	0.963
FastRP (0% Features)	0.964	0.96	0.96	0.96	0.964
FastRP (50% Features)	0.885	0.9	0.89	0.88	0.885
FastRP (100% Features)	0.883	0.9	0.88	0.88	0.883
GraphSAGE (Mean Aggregator)	0.916	0.92	0.92	0.92	0.916
GraphSAGE (Pooling Aggregator)	0.643	0.64	0.64	0.64	0.643
HashGNN (0% Features)	0.674	0.68	0.67	0.67	0.674
HashGNN (50% Features)	0.572	0.6	0.57	0.54	0.572

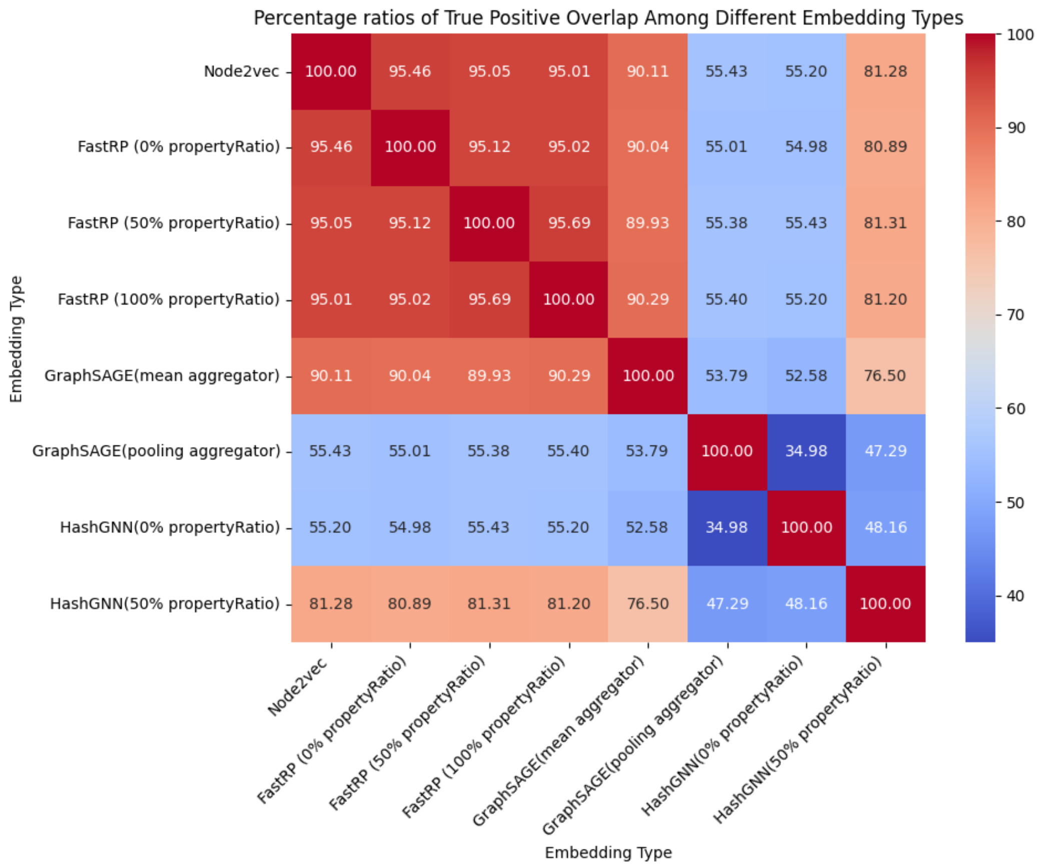
In the case of ego-Facebook dataset, we observed that the Logistic Regression model performed better while utilizing the embedding types of Node2vec, FastRP and GraphSAGE (mean aggregator). Furthermore, algorithms that incorporate the use of adjustable feature levels (e.g. FastRP, HashGNN) exhibited a decline in performance, as the degree of feature involvement increased, with HashGNN (with propertyRatio = 0.5) exhibiting almost complete lack of discriminative ability. Based on these results, we derived that in the context of the ego-Facebook dataset, a topological approach where embeddings are entirely generated using graph structure yields optimal results.

Table 8: Logistic Regression True Positives for different embedding types, regarding the ego-Facebook dataset.



	True Positives	True Positives (Percentage)
Algorithm		
Node2vec	8193	97.292%
FastRP (0% Features)	8129	96.532%
FastRP (50% Features)	8157	96.865%
FastRP (100% Features)	8133	96.58%
GraphSAGE (Mean Aggregator)	7660	90.963%
GraphSAGE (Pooling Aggregator)	4734	56.217%
HashGNN (0% Features)	4718	56.027%
HashGNN (50% Features)	6988	82.983%

The embedding types Node2vec, FastRP and GraphSAGE (mean aggregator) that displayed superior performance metrics, identified a significant portion of the true positives. Notably, the model utilizing HashGNN emeddings (with propertyRatio = 0.5), displayed poor performance, as indicated by the corresponding metrics, however it predicted over 80% of the true positives correctly. The fact that the HashGNN (with propertyRatio = 0.5) model predicted a significant portion of the true positives correctly, accompanied by its overall poor performance indicates the model's inability to identify the negative class instances.

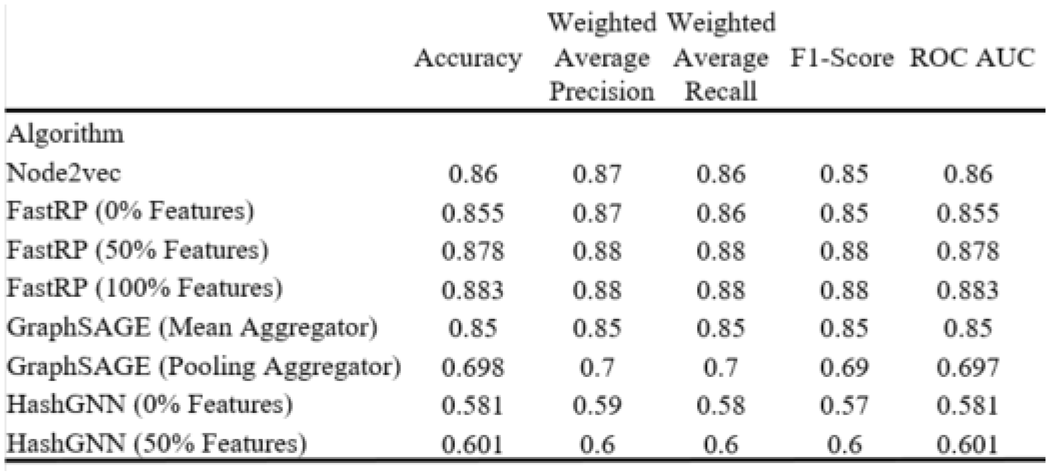


	Node2vec	FastRP (0% propertyRatio)	FastRP (50% propertyRatio)	FastRP (100% propertyRatio)	GraphSAGE(mean aggregator)	GraphSAGE(pooling aggregator)	HashGNN(0% propertyRatio)	HashGNN(50% propertyRatio)
Node2vec	100.00	95.46	95.05	95.01	90.11	55.43	55.20	81.28
FastRP (0% propertyRatio)	95.46	100.00	95.12	95.02	90.04	55.01	54.98	80.89
FastRP (50% propertyRatio)	95.05	95.12	100.00	95.69	89.93	55.38	55.43	81.31
FastRP (100% propertyRatio)	95.01	95.02	95.69	100.00	90.29	55.40	55.20	81.20
GraphSAGE(mean aggregator)	90.11	90.04	89.93	90.29	100.00	53.79	52.58	76.50
GraphSAGE(pooling aggregator)	55.43	55.01	55.38	55.40	53.79	100.00	34.98	47.29
HashGNN(0% propertyRatio)	55.20	54.98	55.43	55.20	52.58	34.98	100.00	48.16
HashGNN(50% propertyRatio)	81.28	80.89	81.31	81.20	76.50	47.29	48.16	100.00

Figure 10: Heatmap of True Positive overlaps between different embedding types, regarding the ego-Facebook dataset. Red bold (100%) cells denote embedding pairs of identical performance predicting true positives, while Blue bold ones (0%) denote no similarity in true positive predictions between embedding pairs.

Feather-LastFM Dataset

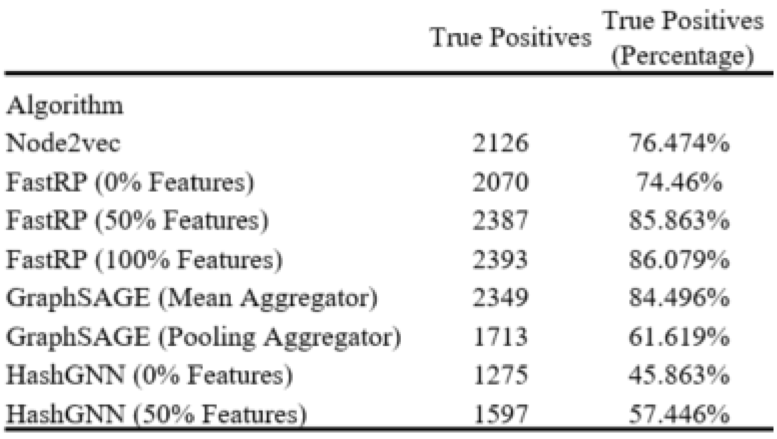
Table 9: Logistic Regression performance evaluation metrics for different embedding types, regarding feather-LastFM dataset.



	Accuracy	Weighted Average Precision	Weighted Average Recall	F1-Score	ROC AUC
Algorithm					
Node2vec	0.86	0.87	0.86	0.85	0.86
FastRP (0% Features)	0.855	0.87	0.86	0.85	0.855
FastRP (50% Features)	0.878	0.88	0.88	0.88	0.878
FastRP (100% Features)	0.883	0.88	0.88	0.88	0.883
GraphSAGE (Mean Aggregator)	0.85	0.85	0.85	0.85	0.85
GraphSAGE (Pooling Aggregator)	0.698	0.7	0.7	0.69	0.697
HashGNN (0% Features)	0.581	0.59	0.58	0.57	0.581
HashGNN (50% Features)	0.601	0.6	0.6	0.6	0.601

Similarly, in the case of the feather-LastFM dataset, we observed that the logistic regression model utilizing the Node2vec, FastRP and GraphSAGE (mean aggregator) embedding types displayed superior performance over those of GraphSAGE (pooling aggregator) and HashGNN. Furthermore, algorithms that incorporate the use of adjustable feature levels (e.g. FastRP, HashGNN) exhibited a slight increase in model performance as the feature involvement increased. Our research findings indicate that incorporating node features in the embedding generation process is the most effective approach, in the context of the feather-LastFM dataset.

Table 10: Logistic Regression True Positives for different embedding types, regarding the feather-LastFM dataset.



	True Positives	True Positives (Percentage)
Algorithm		
Node2vec	2126	76.474%
FastRP (0% Features)	2070	74.46%
FastRP (50% Features)	2387	85.863%
FastRP (100% Features)	2393	86.079%
GraphSAGE (Mean Aggregator)	2349	84.496%
GraphSAGE (Pooling Aggregator)	1713	61.619%
HashGNN (0% Features)	1275	45.863%
HashGNN (50% Features)	1597	57.446%

The research conclusion that we stated is also supported by the fact that the logistic regression model was capable of identifying a significant amount of the true positives correctly, while utilizing embeddings that captured node feature information, with FastRP (with propertyRatio = 0.5, propertyRatio = 1.0) and GraphSAGE (mean aggregator) displaying the best results.

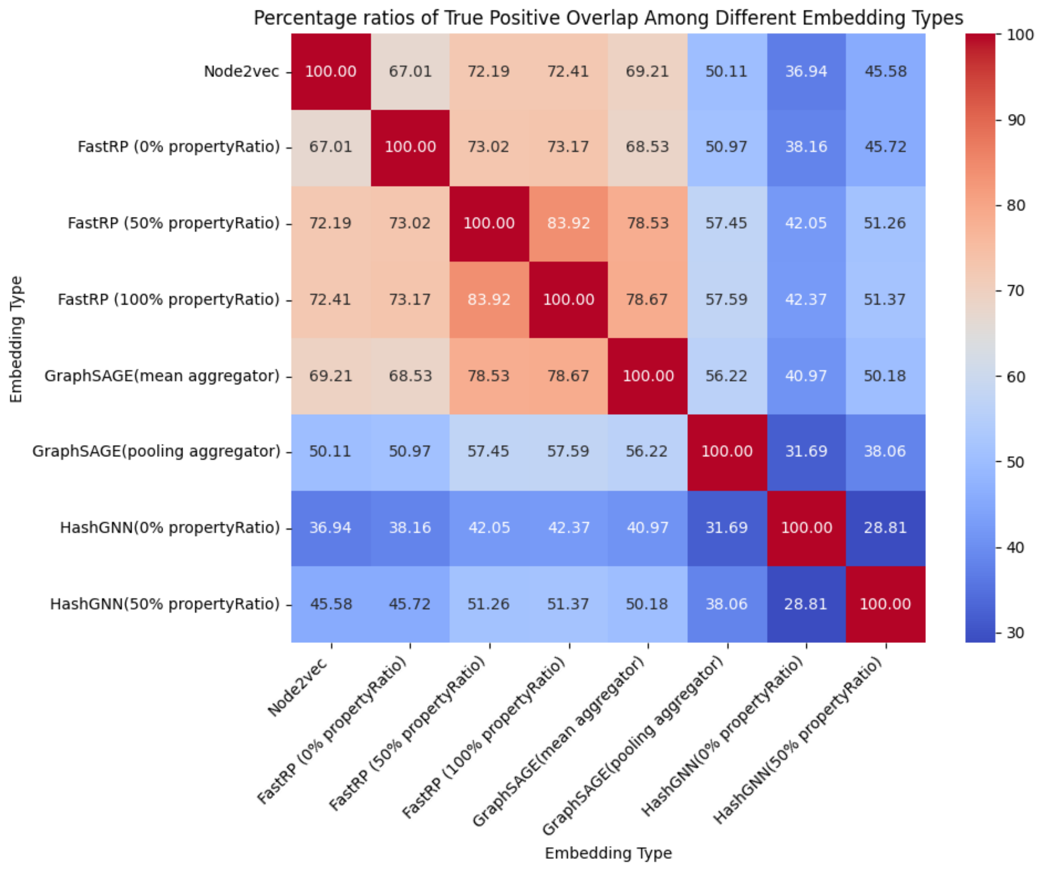
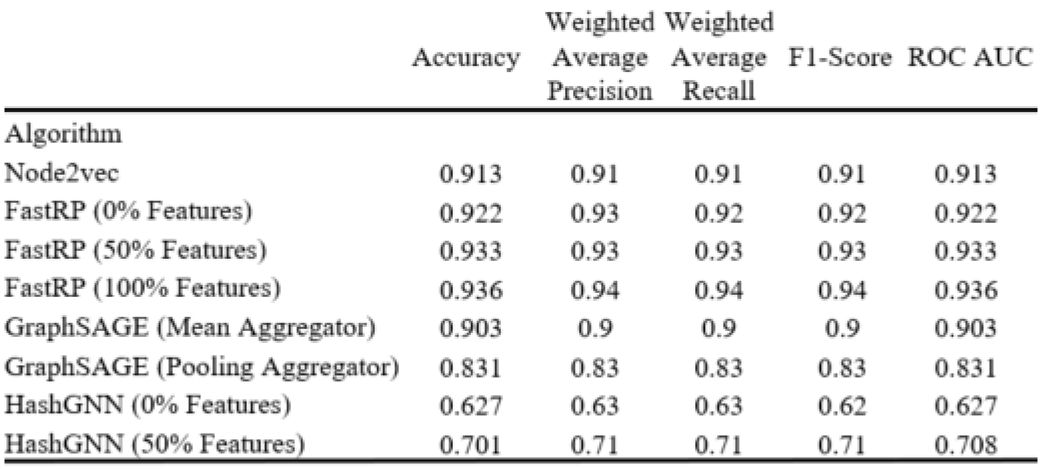


Figure 11: Heatmap of True Positive overlaps between different embedding types, regarding the feather-LastFM dataset. Red bold (100%) cells denote embedding pairs of identical performance predicting true positives, while Blue bold ones (0%) denote no similarity in true positive predictions between embedding pairs.

Musae-Facebook Dataset

Table 11: Logistic Regression performance evaluation metrics for different embedding types, regarding musae-Facebook dataset.



	Accuracy	Weighted Average Precision	Weighted Average Recall	F1-Score	ROC AUC
Algorithm					
Node2vec	0.913	0.91	0.91	0.91	0.913
FastRP (0% Features)	0.922	0.93	0.92	0.92	0.922
FastRP (50% Features)	0.933	0.93	0.93	0.93	0.933
FastRP (100% Features)	0.936	0.94	0.94	0.94	0.936
GraphSAGE (Mean Aggregator)	0.903	0.9	0.9	0.9	0.903
GraphSAGE (Pooling Aggregator)	0.831	0.83	0.83	0.83	0.831
HashGNN (0% Features)	0.627	0.63	0.63	0.62	0.627
HashGNN (50% Features)	0.701	0.71	0.71	0.71	0.708

Musae-Facebook dataset represents another dataset instance, where the involvement of node features in the embeddings improved the model performance, with FastRP displaying the highest performance, followed by Node2vec and GraphSAGE. Our findings suggest that in the context of this specific dataset, embeddings solely based on graph topology or node feature information, produce high quality prediction results. Additionally, this dataset case marks a distinct instance where GraphSAGE (pooling aggregator) demonstrated quality performance results.

Table 12: Logistic Regression True Positives for different embedding types, regarding the musae-Facebook dataset.

Algorithm	True Positives	True Positives (Percentage)
Node2vec	15035	87.924%
FastRP (0% Features)	14929	87.304%
FastRP (50% Features)	15751	92.111%
FastRP (100% Features)	15799	92.391%
GraphSAGE (Mean Aggregator)	15369	89.877%
GraphSAGE (Pooling Aggregator)	13453	78.673%
HashGNN (0% Features)	9320	54.503%
HashGNN (50% Features)	11770	68.830%

The true positives predicted by the model utilizing different embedding types in this case, align with the corresponding performance metrics, with the number of true positives predicted scaling accordingly with each individual embedding type performance.

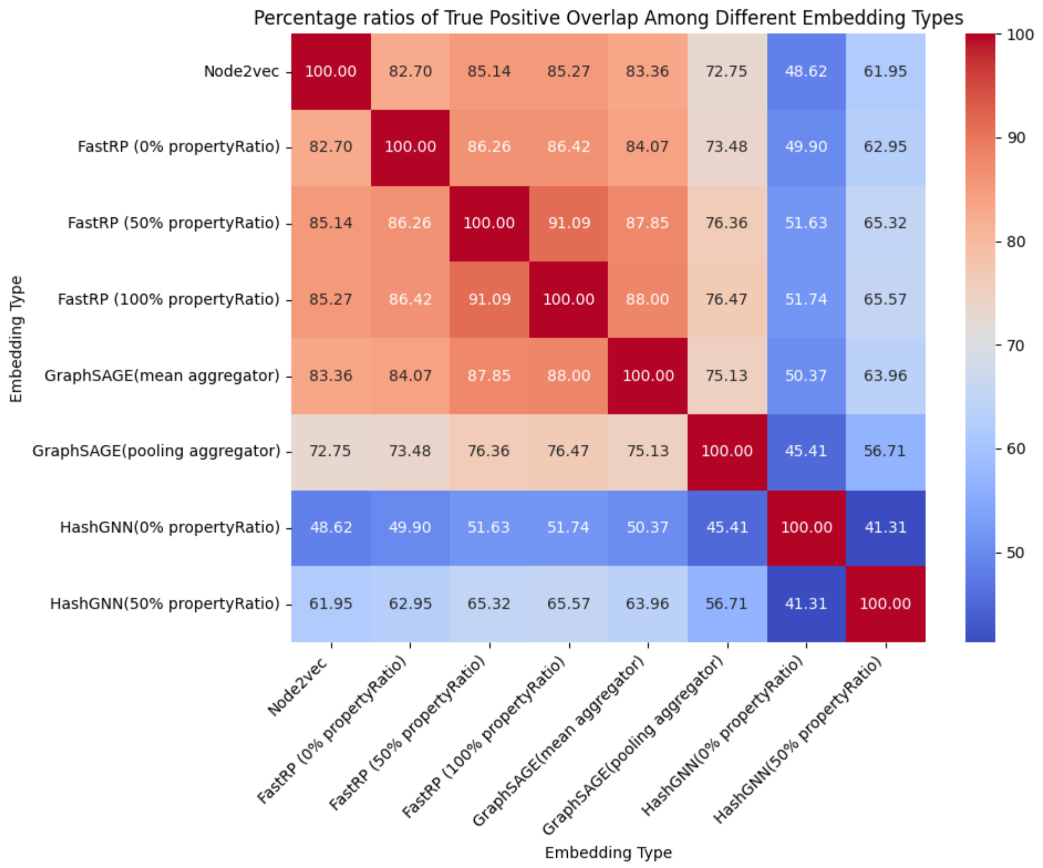
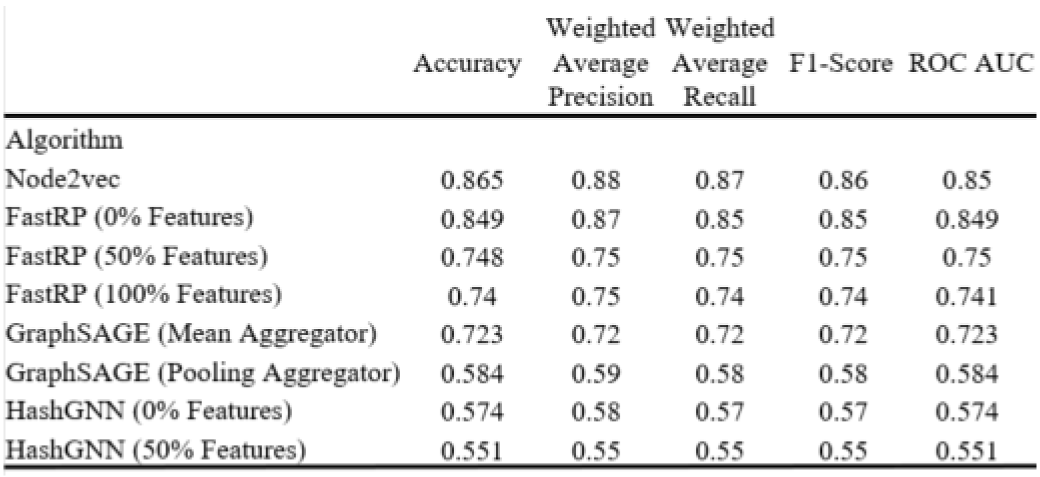


Figure 12: Heatmap of True Positive overlaps between different embedding types, regarding the musae-Facebook dataset. Red bold (100%) cells denote embedding pairs

of identical performance predicting true positives, while Blue bold ones (0%) denote no similarity in true positive predictions between embedding pairs.

Gemsec-Deezer (Croatia) Dataset

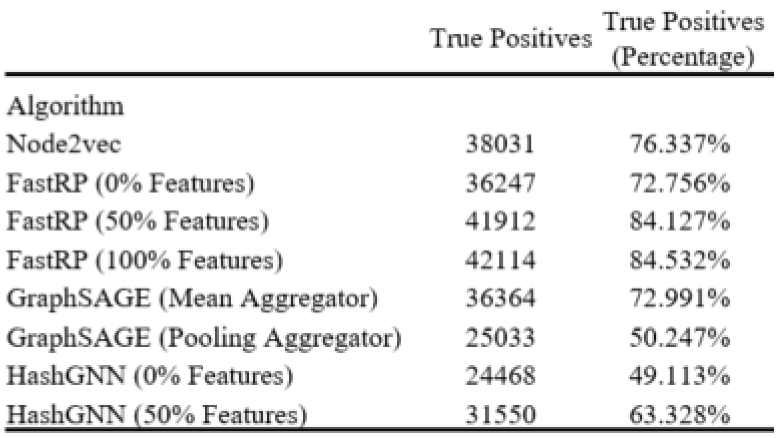
Table 13: Logistic Regression performance evaluation metrics for different embedding types, regarding the gemsec-Deezer (Croatia) dataset.



	Accuracy	Weighted Average Precision	Weighted Average Recall	F1-Score	ROC AUC
Algorithm					
Node2vec	0.865	0.88	0.87	0.86	0.85
FastRP (0% Features)	0.849	0.87	0.85	0.85	0.849
FastRP (50% Features)	0.748	0.75	0.75	0.75	0.75
FastRP (100% Features)	0.74	0.75	0.74	0.74	0.741
GraphSAGE (Mean Aggregator)	0.723	0.72	0.72	0.72	0.723
GraphSAGE (Pooling Aggregator)	0.584	0.59	0.58	0.58	0.584
HashGNN (0% Features)	0.574	0.58	0.57	0.57	0.574
HashGNN (50% Features)	0.551	0.55	0.55	0.55	0.551

We specifically selected this dataset due to its substantial size, to observe the potential impact that an increase in dataset size might have on each distinct embedding performance. Although we observed a decline in performance across all embedding types, Node2vec and FastRP (with propertyRatio = 0.0) remained the most consistent ones. Furthermore, the introduction of node feature involvement in the embeddings displayed a noticeable decline in model performance, with GraphSAGE (pooling aggregator) and HashGNN exhibiting almost complete inability to distinguish between the two classes. Our research findings suggest that an approach incorporating solely graph structure information yields optimal performance results.

Table 14: Logistic Regression True Positives for different embedding types, regarding the gemsec-Deezer (Croatia) dataset.



	True Positives	True Positives (Percentage)
Algorithm		
Node2vec	38031	76.337%
FastRP (0% Features)	36247	72.756%
FastRP (50% Features)	41912	84.127%
FastRP (100% Features)	42114	84.532%
GraphSAGE (Mean Aggregator)	36364	72.991%
GraphSAGE (Pooling Aggregator)	25033	50.247%
HashGNN (0% Features)	24468	49.113%
HashGNN (50% Features)	31550	63.328%

Notably, the model employing the FastRP (with propertyRatio = 0.5, propertyRatio = 1.0) identified over 80% of the true positive instances, a percentage much higher compared to the model of FastRP (with propertyRatio = 0.0). Even though the model using FastRP embeddings, that captured no node feature information, exhibited higher overall performance. This indicates that while the inclusion of node features improved the model's ability to identify true positives, it resulted in a decrease of the model's ability to identify negative instances correctly.

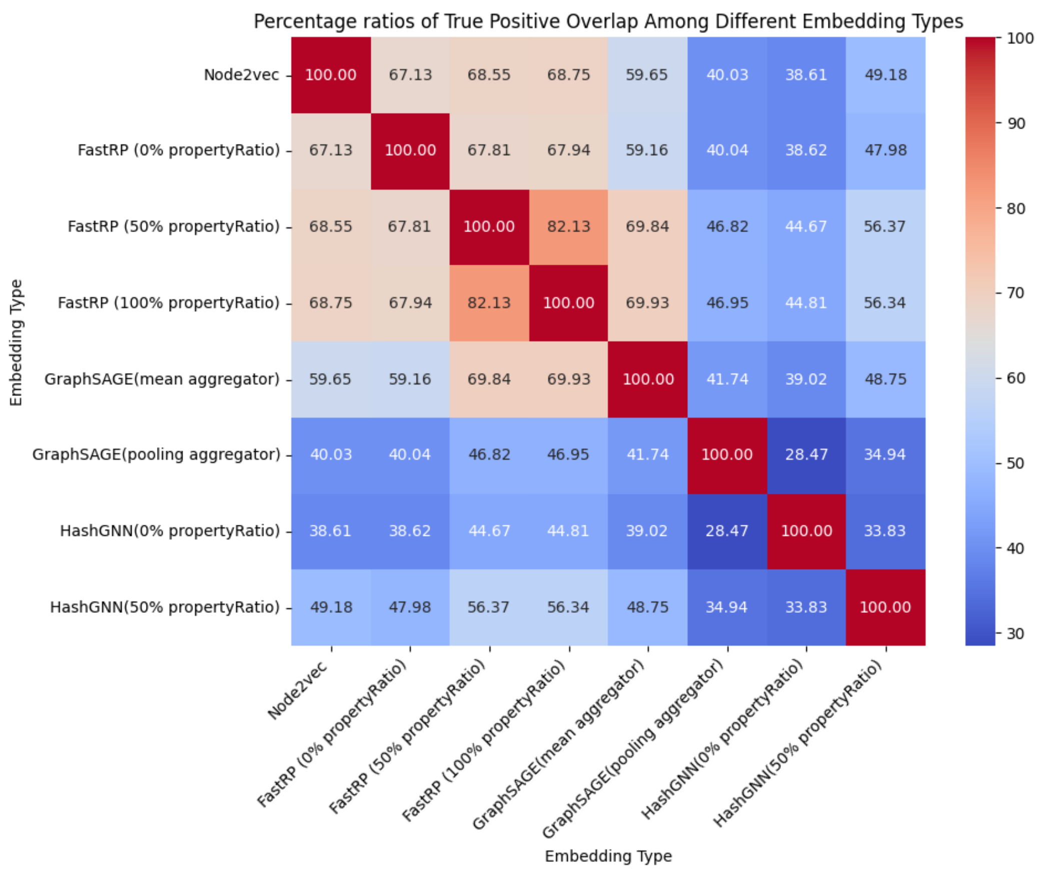
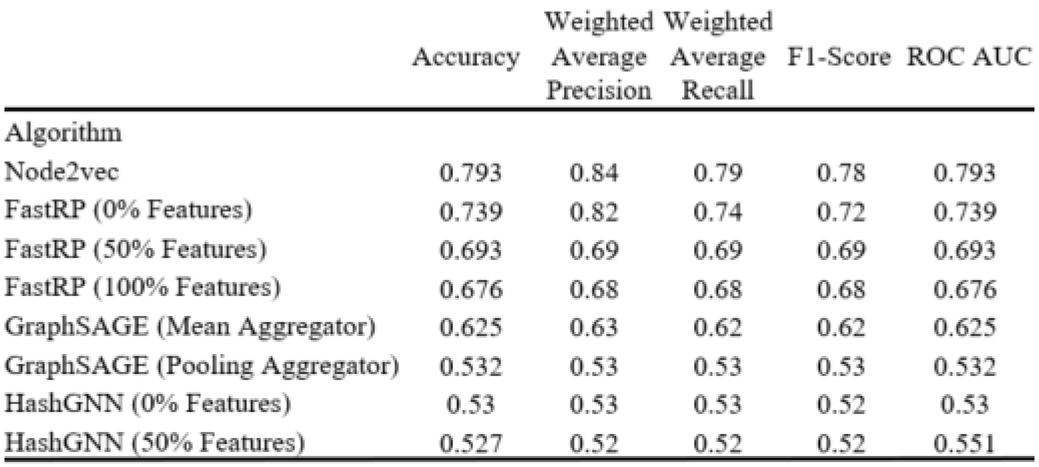


Figure 13: Heatmap of True Positive overlaps between different embedding types, regarding the gemsec-Deezer (Croatia) dataset. Red bold (100%) cells denote embedding pairs of identical performance predicting true positives, while Blue bold ones (0%) denote no similarity in true positive predictions between embedding pairs.

<u>Gemsec-Deezer (Hungary) Dataset</u>

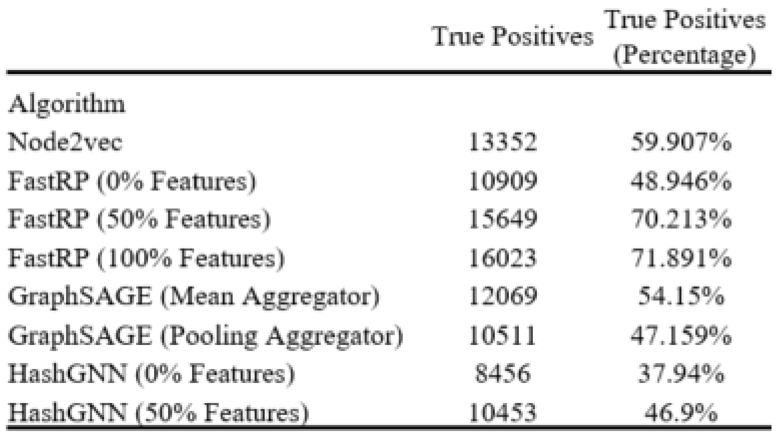
Table 15: Logistic Regression performance evaluation metrics for different embedding types, regarding the gemsec-Deezer (Hungary) dataset.



	Accuracy	Weighted Average Precision	Weighted Average Recall	F1-Score	ROC AUC
Algorithm					
Node2vec	0.793	0.84	0.79	0.78	0.793
FastRP (0% Features)	0.739	0.82	0.74	0.72	0.739
FastRP (50% Features)	0.693	0.69	0.69	0.69	0.693
FastRP (100% Features)	0.676	0.68	0.68	0.68	0.676
GraphSAGE (Mean Aggregator)	0.625	0.63	0.62	0.62	0.625
GraphSAGE (Pooling Aggregator)	0.532	0.53	0.53	0.53	0.532
HashGNN (0% Features)	0.53	0.53	0.53	0.52	0.53
HashGNN (50% Features)	0.527	0.52	0.52	0.52	0.551

This dataset instance consists of a substantial number of graph nodes, comparable to that of the gemsec-Deezer (Croatia), while exhibiting approximately half the number of graph edges. Potentially this difference in the number of edges, between two comparably large graphs, contributed to a performance decline among all distinct embedding types. We observed that although that decline in overall embedding performance occurred, Node2vec and FastRP (with propertyRatio = 0.0) exhibited consistency in producing the highest performance metrics. Evidently, an approach incorporating solely graph structure information yields optimal performance results.

Table 16: Logistic Regression True Positives for different embedding types, regarding the gemsec-Deezer (Hungary) dataset.



	True Positives	True Positives (Percentage)
Algorithm		
Node2vec	13352	59.907%
FastRP (0% Features)	10909	48.946%
FastRP (50% Features)	15649	70.213%
FastRP (100% Features)	16023	71.891%
GraphSAGE (Mean Aggregator)	12069	54.15%
GraphSAGE (Pooling Aggregator)	10511	47.159%
HashGNN (0% Features)	8456	37.94%
HashGNN (50% Features)	10453	46.9%

Identically to the gemsec-Deezer (Croatia) dataset, while the topological approach of Node2vec and FastRP (with propertyRatio = 0.0) provide the highest performance metrics, they cannot identify true positives correctly. The inclusion of node features improves the model's ability to identify true positives as stated by the embedding instances of FastRP (with propertyRatio = 0.5 and propertyRatio = 1.0), however results in a decrease of the model's ability to identify negative instances.

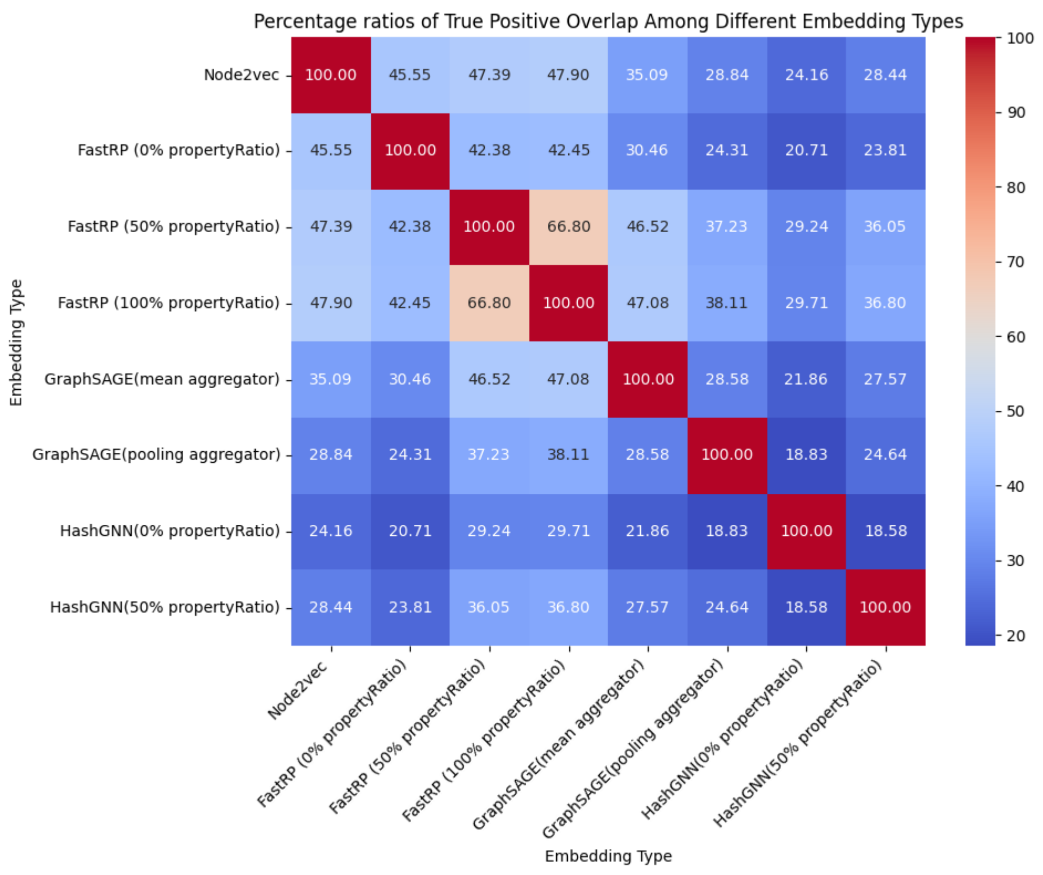


Figure 14: Heatmap of True Positive overlaps between different embedding types, regarding the gemsec-Deezer (Hungary) dataset. Red bold (100%) cells denote embedding pairs of identical performance predicting true positives, while Blue bold ones (0%) denote no similarity in true positive predictions between embedding pairs.

Our experimental evaluation suggests that consistently the Node2vec and FastRP algorithms provide the most prominent results across all selected dataset cases, with the involvement of node features in the embedding generation process, occasionally enhancing or diminishing the model's discriminative ability.

4.6 Predicted Graph Creation

In phase 4.5 of our research, we approached the evaluation of distinct embedding types utilizing various performance metrics. Building on that same principle, we constructed predicted graphs, by adding the predictions made by the trained logistic regression model, for each individual embedding type, to the train graph. We opted for a number of 100 predictions, over the probability threshold of 0.5, from a sorted set of test graph edges. The unique mechanisms in which different embedding algorithms approach the capturing of structural and feature information, result in different predicted edges. Evidently, each embedding case omitted different test graph edges, resulting in the creation of distinct predicted graphs. Based on these predicted graph distinctions, we anticipate different query result errors, in comparison to the ground truth.

Chapter 5. Query Selection

In the framework of Neo4j, we utilized the Cypher query language to develop queries adjusted to context of each individual dataset. Neo4j inherently limits the amount of streamed elements for a single query output. Therefore, we constructed queries returning the count of relationship instances, having common features. We selected these queries, to reach logical conclusions, regarding these datasets. Specifically, the impact each feature might have on the formation of new links. We present the set of queries that we have developed for each specific dataset case:

Ego-Facebook Dataset

- Query returning the number of relationships between people, based on the gender feature:

```
MATCH (p1:Person)-[:FRIENDS_WITH]-(p2:Person)
RETURN
p1.gender1 AS Gender1_p1,
p1.gender2 AS Gender2_p1,
p2.gender1 AS Gender1_p2,
p2.gender2 AS Gender2_p2,
COUNT(*) AS relCount
ORDER BY relCount DESC;
```

In this dataset, some individuals have not designated their gender, i.e., the gender feature is non-binary. Therefore, the gender feature is represented using both node attributes gender1, gender2.

- Query returning the number of relationships between people, based on education concentration feature:

```
MATCH (p1:Person)-[:FRIENDS_WITH]-(p2:Person)
RETURN
p1.education_concentration1 AS EducationConcentration1_p1,
p2.education_concentration1 AS EducationConcentration1_p2,
COUNT(*) AS relCount
ORDER BY relCount DESC;
```

- Query returning the number of relationships between people, based on education type feature:

```
MATCH (p1:Person)-[:FRIENDS_WITH]-(p2:Person)
RETURN
p1.education_type1 AS EducationType1_p1,
p1.education_type2 AS EducationType2_p1,
p1.education_type3 AS EducationType3_p1,
p2.education_type1 AS EducationType1_p2,
p2.education_type2 AS EducationType2_p2,
p2.education_type3 AS EducationType3_p2,
COUNT(*) AS relCount
ORDER BY relCount DESC;
```

We opted for queries that isolate and yield relationship results based on specific node attributes. Utilizing these specific queries, we aim to gain insight into how each feature influences the creation of friendships between people in the social network of Facebook. Additionally, more complex queries can be synthesized combining these ones, aiming to reveal intricate feature patterns and their influence on mutual friendships.

<u>**Feather-LastFM Dataset**</u>

- Query returning the number of relationships between people of the same country

```
MATCH (p1:Person)-[:FRIENDS_WITH]-(p2:Person)
WHERE p1.country = p2.country AND id(p1) < id(p2)
RETURN p1.country AS country,
COUNT(*) AS relCount
ORDER BY relCount DESC;
```

- Query returning the number of relationships between people of different countries

```
MATCH (p1:Person)-[:FRIENDS_WITH]-(p2:Person)
WHERE p1.country <> p2.country AND id(p1) < id(p2)
RETURN p1.country AS country1, p2.country AS country2,
COUNT(*) AS relCount
ORDER BY relCount DESC;
```

In the context of this dataset, we utilized these specific queries, aiming to gain insight into how nationality influences the creation of friendships between people in the social network of LastFM Asia. We attempted to incorporate the feature vectors that represent the artists liked by each individual, into the queries regarding nationality. Successfully combining the nationality and liked artists into unified queries would allow us to provide more conclusive results of the balance between nationality and music preference and its effects on friendships, in the context of the social network LastFM.

Musae-Facebook Dataset

- Query returning the number of likes between pages of the same category type

```
MATCH (p1:Page)-[:LIKES]-(p2:Page)
WHERE p1.type = p2.type
WITH p1.type AS type,
COUNT(*) AS relCount
RETURN type, relCount
ORDER BY relCount DESC;
```

- Query returning the number of likes between pages of different category types

```
MATCH (p1:Page)-[:LIKES]-(p2:Page)
WHERE p1.type <> p2.type
WITH p1.type AS type1, p2.type AS type2,
COUNT(*) AS relCount
RETURN type1, type2, relCount
ORDER BY relCount DESC;
```

The musae-Facebook large dataset, represents mutual likes between verified pages of specific categories. These queries present information regarding the influence of page categories and the likelihood of mutually liking each other. For instance, in the context of the categories politician and government, a mutual like could potentially indicate alignment of identical political beliefs and ideologies.

Gemsec-Deezer (Croatia, Hungary)

- Query returning the number of mutual friendships between people who share a certain amount of liked genres or more

```
MATCH (p1:Person)-[:FRIENDS_WITH]-(p2:Person)
WITH p1, p2,
[i IN range(0, size(p1.features) - 1) | CASE WHEN p1.features[i] = 1 AND p2.features[i] = 1 THEN 1 ELSE 0 END] AS commonGenres,
WITH reduce(s = 0, x IN commonGenres | s + x) AS commonGenresCount
WHERE commonGenresCount > 0
COUNT(*) AS relCount
RETURN commonGenresCount, relCount
ORDER BY commonGenresCount DESC;
```

Utilizing this specific query yields the number of mutual friendships that share a number of common interests over a threshold. Higher commonGenresCount values or adjustments to fixed thresholds, provide information focused on how an increase in shared preferences affect the creation of friendships, in the context of the social network Deezer.

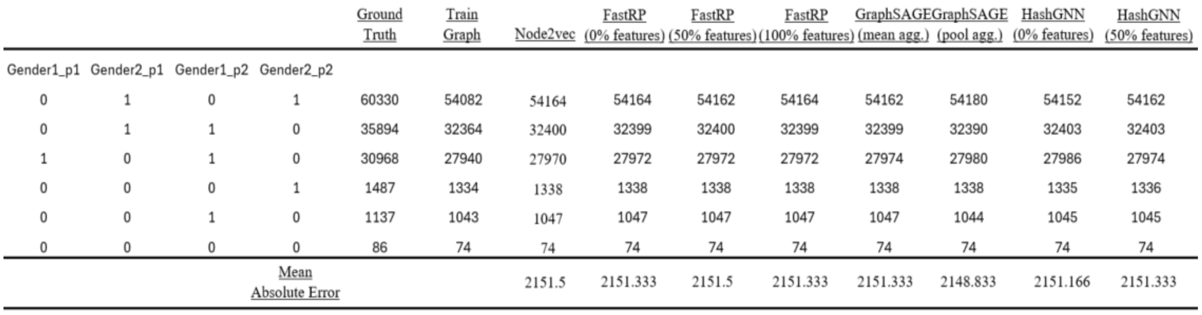
Chapter 6. Query Error Calculation

In the previous section, we defined the queries that we employed to gain insight into the selected datasets, as well as evaluate further the influence of different embedding algorithms in the research results. The queries that we utilized, yield edge instances, classifying them into distinct feature categories. In order to calculate this impact for each distinct graph case, we utilized the Mean Absolute Error (MAE) metric, that is defined as:

$$Mean\ Absolute\ Error = \frac{\sum_{i=1}^{n} |Query\_Result_{Ground\_Truth_i} - Query\_Result_{Predicted_i}|}{n}.$$

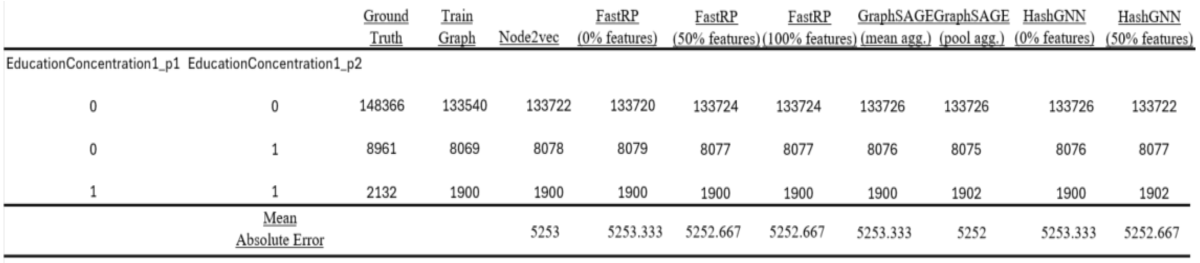
During the data preprocessing phase, we removed edges from the ground truth, resulting into two separate training and test graphs. The edges were removed non-uniformly from the various feature categories. Similarly, the edges that were predicted were also not added uniformly into these distinct categories. We expect that these factors will produce different degrees of MAE, in the query results of each individual predicted graph.

Table 17: Predicted graph mean absolute error, regarding relationships based on gender, for the ego-Facebook dataset.



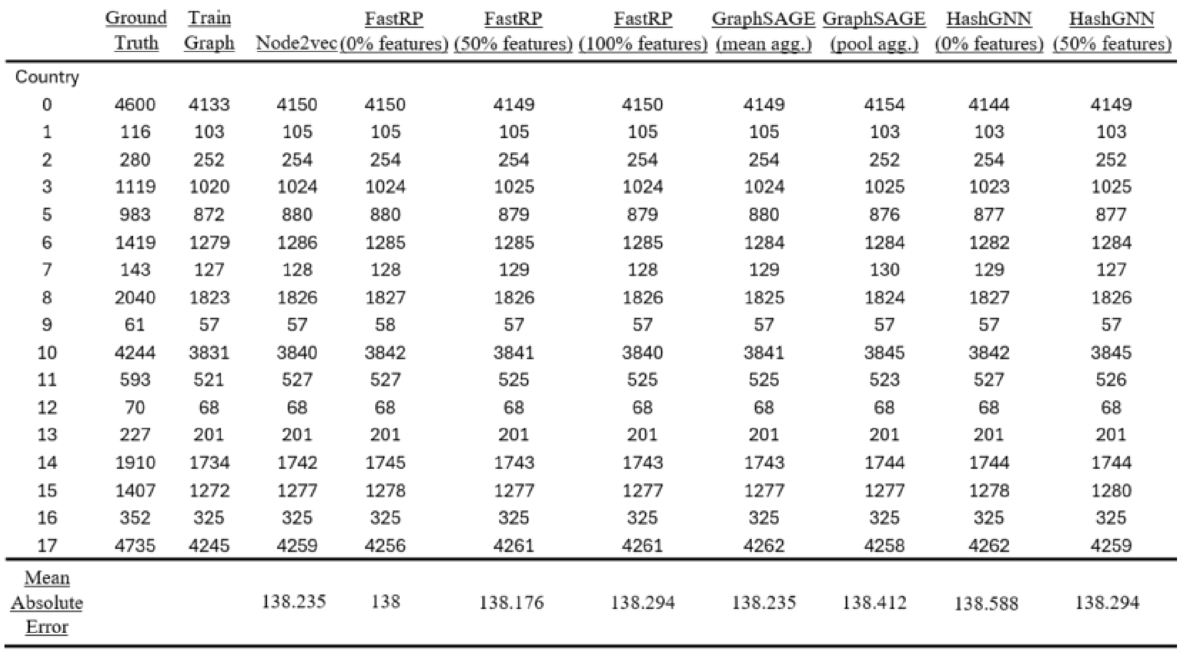
Gender1_p1	Gender2_p1	Gender1_p2	Gender2_p2	Ground Truth	Train Graph	Node2vec	FastRP (0% features)	FastRP (50% features)	FastRP (100% features)	GraphSAGE (mean agg.)	GraphSAGE (pool agg.)	HashGNN (0% features)	HashGNN (50% features)
0	1	0	1	60330	54082	54164	54164	54162	54164	54162	54180	54152	54162
0	1	1	0	35894	32364	32400	32399	32400	32399	32399	32390	32403	32403
1	0	1	0	30968	27940	27970	27972	27972	27972	27974	27980	27986	27974
0	0	0	1	1487	1334	1338	1338	1338	1338	1338	1338	1335	1336
0	0	1	0	1137	1043	1047	1047	1047	1047	1047	1044	1045	1045
0	0	0	0	86	74	74	74	74	74	74	74	74	74
			Mean Absolute Error			2151.5	2151.333	2151.5	2151.333	2151.333	2148.833	2151.166	2151.333

Table 18: Predicted graph mean absolute error, regarding relationships based on education concentration, for the ego-Facebook dataset.



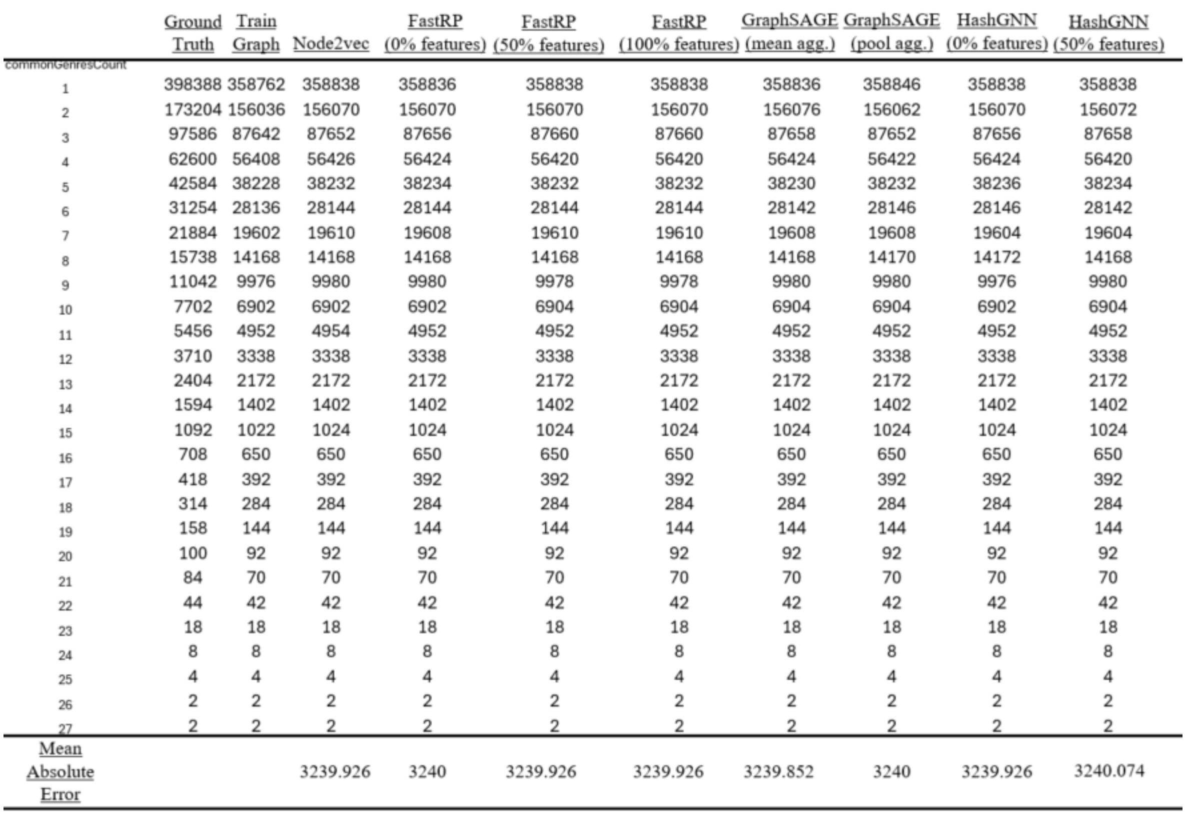
EducationConcentration1_p1	EducationConcentration1_p2	Ground Truth	Train Graph	Node2vec	FastRP (0% features)	FastRP (50% features)	FastRP (100% features)	GraphSAGE (mean agg.)	GraphSAGE (pool agg.)	HashGNN (0% features)	HashGNN (50% features)
0	0	148366	133540	133722	133720	133724	133724	133726	133726	133726	133722
0	1	8961	8069	8078	8079	8077	8077	8076	8075	8076	8077
1	1	2132	1900	1900	1900	1900	1900	1900	1902	1900	1902
	Mean Absolute Error			5253	5253.333	5252.667	5252.667	5253.333	5252	5253.333	5252.667

Table 19: Predicted graph mean absolute error, regarding relationships of the same country, for the feather-LastFM dataset.



	Ground Truth	Train Graph	Node2vec	FastRP (0% features)	FastRP (50% features)	FastRP (100% features)	GraphSAGE (mean agg.)	GraphSAGE (pool agg.)	HashGNN (0% features)	HashGNN (50% features)
Country										
0	4600	4133	4150	4150	4149	4150	4149	4154	4144	4149
1	116	103	105	105	105	105	105	103	103	103
2	280	252	254	254	254	254	254	252	254	252
3	1119	1020	1024	1024	1025	1024	1024	1025	1023	1025
5	983	872	880	880	879	879	880	876	877	877
6	1419	1279	1286	1285	1285	1285	1284	1284	1282	1284
7	143	127	128	128	129	128	129	130	129	127
8	2040	1823	1826	1827	1826	1826	1825	1824	1827	1826
9	61	57	57	58	57	57	57	57	57	57
10	4244	3831	3840	3842	3841	3840	3841	3845	3842	3845
11	593	521	527	527	525	525	525	523	527	526
12	70	68	68	68	68	68	68	68	68	68
13	227	201	201	201	201	201	201	201	201	201
14	1910	1734	1742	1745	1743	1743	1743	1744	1744	1744
15	1407	1272	1277	1278	1277	1277	1277	1277	1278	1280
16	352	325	325	325	325	325	325	325	325	325
17	4735	4245	4259	4256	4261	4261	4262	4258	4262	4259
Mean Absolute Error			138.235	138	138.176	138.294	138.235	138.412	138.588	138.294

Table 20: Predicted graph mean absolute error, regarding relationships sharing common music interests above a specific threshold, for the gemsec-Deezer (Croatia) dataset.



commonGenresCount	Ground Truth	Train Graph	Node2vec	FastRP (0% features)	FastRP (50% features)	FastRP (100% features)	GraphSAGE (mean agg.)	GraphSAGE (pool agg.)	HashGNN (0% features)	HashGNN (50% features)
1	398388	358762	358838	358836	358838	358838	358836	358846	358838	358838
2	173204	156036	156070	156070	156070	156070	156076	156062	156070	156072
3	97586	87642	87652	87656	87660	87660	87658	87652	87656	87658
4	62600	56408	56426	56424	56420	56420	56424	56422	56424	56420
5	42584	38228	38232	38234	38232	38232	38230	38232	38236	38234
6	31254	28136	28144	28144	28144	28144	28142	28146	28146	28142
7	21884	19602	19610	19608	19610	19610	19608	19608	19604	19604
8	15738	14168	14168	14168	14168	14168	14168	14170	14172	14168
9	11042	9976	9980	9980	9978	9978	9980	9980	9976	9980
10	7702	6902	6902	6902	6904	6904	6904	6904	6902	6904
11	5456	4952	4954	4952	4952	4952	4952	4952	4952	4952
12	3710	3338	3338	3338	3338	3338	3338	3338	3338	3338
13	2404	2172	2172	2172	2172	2172	2172	2172	2172	2172
14	1594	1402	1402	1402	1402	1402	1402	1402	1402	1402
15	1092	1022	1024	1024	1024	1024	1024	1024	1024	1024
16	708	650	650	650	650	650	650	650	650	650
17	418	392	392	392	392	392	392	392	392	392
18	314	284	284	284	284	284	284	284	284	284
19	158	144	144	144	144	144	144	144	144	144
20	100	92	92	92	92	92	92	92	92	92
21	84	70	70	70	70	70	70	70	70	70
22	44	42	42	42	42	42	42	42	42	42
23	18	18	18	18	18	18	18	18	18	18
24	8	8	8	8	8	8	8	8	8	8
25	4	4	4	4	4	4	4	4	4	4
26	2	2	2	2	2	2	2	2	2	2
27	2	2	2	2	2	2	2	2	2	2
Mean Absolute Error			3239.926	3240	3239.926	3239.926	3239.852	3240	3239.926	3240.074

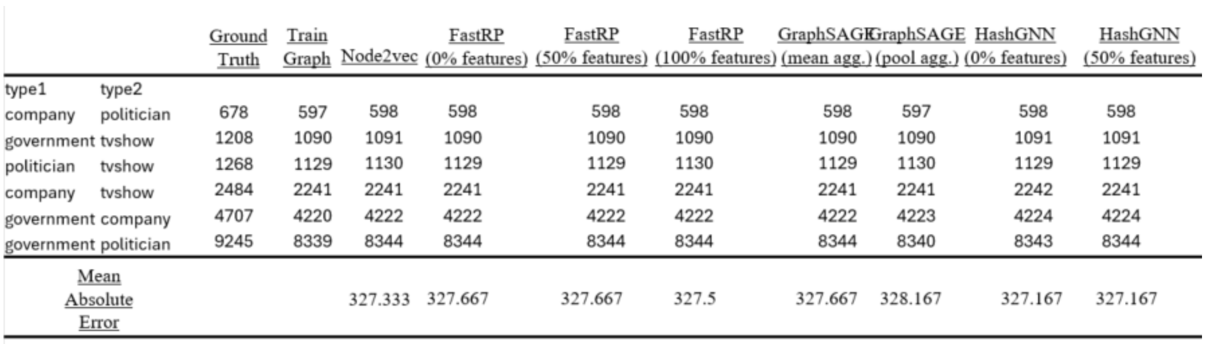
Table 21: Predicted graph mean absolute error, regarding relationships sharing common music interests above a specific threshold, for the gemsec-Deezer (Hungary) dataset.

	Ground Truth	Train Graph	Node2vec	FastRP (0% features)	FastRP (50% features)	FastRP (100% features)	GraphSAGE (mean agg.)	GraphSAGE (pool agg.)	HashGNN (0% features)	HashGNN (50% features)
commonGenresCount										
1	173044	155892	155950	155952	155958	155966	155960	155954	155958	155966
2	73154	65602	65640	65646	65630	65628	65632	65644	65642	65632
3	42494	38236	38260	38252	38256	38256	38258	38262	38252	38254
4	27148	24338	24352	24346	24352	24350	24348	24352	24356	24352
5	18368	16536	16542	16544	16548	16548	16550	16542	16542	16544
6	13008	11694	11704	11700	11702	11702	11706	11698	11698	11704
7	9080	8166	8166	8168	8170	8168	8168	8166	8168	8170
8	6324	5680	5682	5682	5682	5682	5682	5684	5680	5684
9	4372	3922	3926	3926	3928	3926	3928	3928	3926	3928
10	2914	2604	2606	2606	2606	2606	2606	2606	2604	2604
11	2086	1886	1886	1886	1886	1886	1886	1886	1886	1886
12	1492	1344	1344	1344	1344	1344	1344	1344	1344	1344
13	1030	940	940	940	940	940	940	940	940	940
14	602	538	538	538	538	538	538	538	538	538
15	394	356	356	356	356	356	356	356	356	356
16	240	218	218	218	218	218	218	218	218	218
17	172	148	148	148	148	148	148	148	148	148
18	82	74	74	74	74	74	74	74	74	74
19	44	38	38	38	38	38	38	38	38	38
20	34	30	30	30	30	30	30	30	30	30
21	26	20	20	20	20	20	20	20	20	20
22	6	6	6	6	6	6	6	6	6	6
23	2	2	2	2	2	2	2	2	2	2
24	10	8	8	8	8	8	8	8	8	8
25	4	4	4	4	4	4	4	4	4	4
26	2	2	2	2	2	2	2	2	2	2
27	2	2	2	2	2	2	2	2	2	2
Mean Absolute Error			1395.926	1396.148	1395.778	1395.778	1395.556	1395.63	1396	1395.556

Table 22: Predicted graph mean absolute error, regarding likes between pages of the same type, for the musae-Facebook dataset.

	Ground Truth	Train Graph	Node2vec	FastRP (0% features)	FastRP (50% features)	FastRP (100% features)	GraphSAGE (mean agg.)	GraphSAGE (pool agg.)	HashGNN (0% features)	HashGNN (50% features)
type										
Company	40179	36236	36258	36252	36256	36256	36254	36250	36254	36246
Government	162707	146182	146286	146292	146290	146290	146292	146292	146308	146294
Politician	73800	66575	66623	66627	66625	66623	66625	66633	66603	66623
Tv Show	25959	23421	23427	23427	23427	23427	23427	23429	23427	23429
Mean Absolute Error			7512.75	7511.75	7511.75	7512.25	7511.75	7510.25	7513.25	7513.25

Table 23: Predicted graph mean absolute error, regarding likes between pages of the different types, for the musae-Facebook dataset.



type1	type2	Ground Truth	Train Graph	Node2vec	FastRP (0% features)	FastRP (50% features)	FastRP (100% features)	GraphSAGE (mean agg.)	GraphSAGE (pool agg.)	HashGNN (0% features)	HashGNN (50% features)
company	politician	678	597	598	598	598	598	598	597	598	598
government	tvshow	1208	1090	1091	1090	1090	1090	1090	1090	1091	1091
politician	tvshow	1268	1129	1130	1129	1129	1130	1129	1130	1129	1129
company	tvshow	2484	2241	2241	2241	2241	2241	2241	2241	2242	2241
government	company	4707	4220	4222	4222	4222	4222	4222	4223	4224	4224
government	politician	9245	8339	8344	8344	8344	8344	8344	8340	8343	8344
Mean Absolute Error				327.333	327.667	327.667	327.5	327.667	328.167	327.167	327.167

Evidently, the mean absolute errors that we observed are different for each algorithm, conclusively proving that the stochastic behavior involved in the splitting of the initial graph and embedding generation processes, influences the produced results and introduces an inherent error factor in the predictions.

Chapter 7. Conclusions and Limitations

In our diploma thesis, we presented the notion that various factors influence the performance of machine learning models. In the course of our research, we incorporated various machine learning models and embedding types. We observed notable differences in the produced results of the same logistic regression model, utilizing various embedding types. Furthermore, different models exhibit variations in performance, when using identical embeddings. Moreover, performance results may potentially be altered by different approaches during the data preprocessing phase. These approaches involve selecting different ways of feature information representation and the stochastic behavior in the process of splitting graphs. We concluded that these various factors introduce an inherent error in the process of machine learning graph analysis, affecting the insights that we attempt to gain for each real-world dataset case.

Considering the findings of our research, we underline the importance of carefully selecting the parameters that influence the process of machine learning, based on the specific requirements of each individual dataset we aim to analyze.

In the context of our research, it is important to note that there were several limitations. Specifically, algorithms such as GraphSAGE and HashGNN could potentially produce optimal results, provided the appropriate computational resources allowing for parameter fine-tuning. Similarly, had we had sufficient computational resources, we could have employed the KNN or lasso classifier models, both of which displayed slightly better prediction accuracy, than the logistic regression model.

References

1. Mcauley, J. and Leskovec, J. 2014. Discovering social circles in ego networks. *ACM Transactions on Knowledge Discovery from Data*. 8, 1 (Feb. 2014), 1–28. DOI: https://doi.org/10.1145/2556612.
2. Rozemberczki, B. and Sarkar, R. 2020. Characteristic Functions on Graphs: Birds of a Feather, from Statistical Descriptors to Parametric Models. *Proceedings of the 29th ACM International Conference on Information & Knowledge Management* (Virtual Event Ireland, Oct. 2020), 1325–1334.
3. Rozemberczki, B., Allen, C., and Sarkar, R. 2021. Multi-Scale Attributed Node Embedding. IMA Journal of Complex Networks, 2021. DOI:10.1093/comnet/xxx000.
4. Rozemberczki, B. et al. 2019. GEMSEC: graph embedding with self clustering. *Proceedings of the 2019 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining* (Vancouver British Columbia Canada, Aug. 2019), 65–72.
5. 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, San Francisco, CA, USA, August 13-17, 2016. ACM, 855–864. DOI: https://doi.org/10.1145/2939672.2939754.
6. Haochen Chen, Syed Fahad Sultan, Yingtao Tian, Muhao Chen, and Steven Skiena. 2019. Fast and Accurate Network Embeddings via Very Sparse Random Projection. In The 28th ACM International Conference on Information and Knowledge Management (CIKM'19), November 3–7, 2019, Beijing, China. ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3357384.3357879
7. Hamilton, W. L., Ying, R., and Leskovec, J. 2018. Inductive Representation Learning on Large Graphs. In Proceedings of the 31st Conference on Neural Information Processing Systems (NeurIPS 2017), Long Beach, CA, USA. arXiv:1706.02216v4 [cs.SI].
8. Qiaoyu Tan, Ninghao Liu, Xing Zhao, Hongxia Yang, Jingren Zhou, and Xia Hu. 2020. Learning to Hash with Graph Neural Networks for Recommender Systems. In Proceedings of The Web Conference 2020 (WWW '20), April 20–24, 2020, Taipei, Taiwan. ACM, New York, NY, USA, 11 pages. https://doi.org/10.48550/arXiv.2003.01917
9. Wei Wu, Bin Li, Chuan Luo, and Wolfgang Nejdl. 2021. Hashing-Accelerated Graph Neural Networks for Link Prediction. In Proceedings of the Web Conference 2021 (WWW '21), April 19–23, 2021, Ljubljana, Slovenia. ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3442381.3449884