OTTO VON GUERICKE UNIVERSITÄT MAGDEBURG INF FACULTY OF COMPUTER SCIENCE

Design and development of a standardized framework for fairness analysis and mitigation for Graph Neural Network-based user profiling models

Master Thesis

Author:
Mohamed Abdelrazek

January 31, 2023

First Examiner: Prof. Dr.-Ing. Ernesto William De Luca

Second Examiner: Dr. Ludovico Boratto

Supervisor: M.Sc. Erasmo Purificato

Institute of Technical and Business Information Systems
Digital Transformation and Digital Humanities

Abdelrazek, Mohamed:
Design and development of a standardised framework for fairness analysis and mitigation for Graph Neural Network-based user profiling models
Master Thesis, Otto-von-Guericke University
Magdeburg, 2023.

Abstract

User profiling has been a very popular problem in the last year in regards to machine learning, which concentrates on classifying users into a specific category. After the introduction of Graph Neural Networks (GNNs) in last years, user profiling has been represented as a node classification problem to better understand and account for the relationship between users. This paved the way to many new state of the art GNN models structures to solve the user profiling problem while concentrating on many different aspects, which gave the user so many options to consider from. Additionally, most of these models only concentrate on evaluating how good is the model prediction, while neglecting the model fairness. In this work we design and develop a novel framework for fairness analysis and mitigation based on user profiling. The framework goal is to allow users to better analyze and compare different user profiling models at the same time, as well as to evaluate model fairness and be able to experiment with different bias mitigation approaches if needed, making it easier for users to choose the best suitable model for them. Since every model requires a different input data type structure, we overcome this problem by designing a standardised pre-processing approach which makes it easier for the user to train several "state-of-the-art GNN models" sequentially using only a single data type structure. To this end, we also conducted a series of preliminary experiments to compare the fairness of several state of the art GNN models using several pre-processing debiasing approaches. By utilizing the disparate impact and disparate mistreatment metrics, we evaluate the fairness of the models and find that certain debiasing methods can lead to fairer GNN predictions in some cases using different datasets.

Acknowledgements

I would like to thank all the people that supported me and made this thesis possible.

First I would like to thank both my examiners firstly Prof. Dr.-Ing. Ernesto William De Luca, for granting me the opportunity to write this thesis at his research group, and secondly Dr. Ludovico Boratto for his collaboration in this work and his insightful knowledge in this domain. Furthermore, I would like to give a very special thanks to my advisor M.Sc. Erasmo Purificato for being my advisor for this thesis, and supporting me whenever I had any questions and problems throughout writting this thesis. Thanks for making the past year very enjoyable.

I also want to thank my friends, for always supporting me from the start, and for giving me fun and refreshing time whenever I was exhausted and stressed out.

Finally, I owe my deepest thanks to my family, since without them I would not have achieved what I achieved in last years.

Contents

List of Figures

List of Tables

List of Algorithms

Aller Anfang ist schwer.

Publius Ovidius Naso

1 Introduction

Many users are creating a huge amount of data everyday while using different online applications. This results in having such huge amount of data from users actions and activities. Additionally this made user profiling become a very interesting topic in the last few years specifically in social media applications. The objective of user profiling is to deduce a person's preferences, characteristics, or actions from the data collected to form a comprehensive user profile, known as a user model, which can be utilized by personalized and adaptive systems. Additionally these different user profiles can be classified into different categories depending on individuals' actions and interactions (implicit user profiling) For example a group of users would be classified into a specific category according to a specific aspect like age, gender or users region. An example where user profiling is popular is personalized systems and recommendation systems. There have been many machine learning approaches conducted for classifying user profiles, especially deep learning approaches [1, 2]. In spite of using deep learning approaches, user profiles can be better represented as graphs, to better model the users behaviour, where the edges depict the connections between users, who are represented by nodes.represent. This can be very useful to model complex users behaviors like user to user interaction or user to item interaction. This gives an advantage for graph deep learning approaches such as Graph Neural Networks (GNNs) in comparison to traditional deep learning methods for user profile classification [3, 4, 5, 6, 7]. GNNs have proven to be successful in representing graph data not only with user profiling but also across various fields, such as recommendation systems [8, 9], and natural language processing [10].

Although these GNNs approaches can be very beneficial for user profiling prediction, an effective way of evaluating and understanding the user profiles predictions is also important. Most of the GNNs approaches consider only the model prediction accuracy, but such procedure is not always effective. Users should also consider other aspects such as model fairness and model explainability. Model fairness and fairness metrics have been an important topic in the last few years [11], since the fact that machine learning models in general are taught to mirror the distribution of training data. Eventually this can incorporate historical bias towards sensitive features. This also applies to graph data using GNNs because of the message passing algorithm that is used by GNNs, which at the end shows that some "state-of-the-art" GNN models can be biased in their predictions [12]. Although there have been models that were published trying to minimize discrimination in user profiling classification using in-processing debaising approaches [13], these in-processing approaches can be hard to replicate on other models, as a result making it hard for other GNN models to mitigate bias in their predictions. All of this has motivated our research questions and challenges for this thesis, which are: (i) given such huge popularity for graph neural networks, can we design and develop a standardised framework for training and analysing different GNNs model results regarding predictions and fairness? (ii) since also model fairness has shown a great importance in the last years, and it was also shown in [12] that GNNs model can contain bias in their predictions, this motivated us to find a better fairness debiasing approach that can be easily implemented and also hopefully effective for most GNN models.

We introduce in this work a novel framework that helps researchers in analyzing user profiling results for several models at once. The first version of this framework consists of three models mainly FairGNN [13], RHGN [6], and CatGCN [7]. The main component of the framework is the pre-processing component, which preprocesses the input data for all models accordingly. The framework supports for user input data both Neo4J [1] and NetworkX [2] formats. While having such a framework that can help in analyzing the results of different models easily at once, we also perform a user study to apply different debaising methods on the available models. The chosen debaising approaches are mainly pre-processing approaches, since those approaches can be easily used without altering the main structure of the models. Furthermore those pre-processing ap-

[1] https://neo4j.com/
[2] https://networkx.org/

proaches introduce some questions and challenges, mainly can such pre-processing approaches mitigate bias in training while also guarantee a high prediction accuracy for the models. Finally, can these approaches offer us better prediction results in comparison to the state of the art GNN fairness debaising model FairGNN.

The first goal of this thesis is to implement and evaluate a novel framework that can train different GNNs models sequentially so that it can help researchers in analyzing user profiling predictions results effectively. This includes building and testing a pre-processing component that can pre-process the proposed data format (Neo4J and NetworkX) for all three models respectively. The second goal of this thesis would be to find suitable pre-processing debaising approaches that can suit the framework structure. These debiaisng approaches would be then implemented in a debaising component in the framework. This component would be responsible for first calculating how fair the dataset is and applying the debiasing approaches if needed. Finally we want to evaluate the framework functionality and the pre-processing component, namely using the framework to train different models using a unified input data (e.g. NetworkX format). Furthermore we conduct our proposed user study and evaluate the proposed debiasing approaches on the models and datasets and compare the fairness metric results to FairGNN which will be trained on the same datasets as well.

To conclude, the main contributions of this work are summarized as follows:

- To design and develop a standardised framework for user profiling prediction and fairness analysis using different GNNs models.
- To perform a preliminary user study on fairness analysis and fairness mitigation using the framework that was developed.

The thesis is structured as follows: We first conduct a literature overview about user profiling, graph neural networks, and fairness in GNNs in chapter 2. In chapter 3 we introduce the preliminaries or the needed background knowledge that will be used later on for the different components of the framework(e.g. models, datasets, and fairness metrics). We later introduce in chapter 4 our framework structure and explain in details how each component is defined in our framework and how every component works with the other. We then define our evaluation settings and discuss the framework experiments that we have done on the presented datasets, as well as analyzing the fairness metrics upon using our proposed debaising approaches in chapter 5. This work is ended with a conclusion and an overview of opportunities for future work in chapter 6.

She said, where d'you wanna go?
How much you wanna risk?

"SOMETHING JUST LIKE THIS",
CHAINSMOKERS AND COLDPLAY

2 Background and Related Work

In this chapter we give an overview of other works that are related to our framework. We first give a literature review on Graph Neural Networks (GNNs) and their different types, and then we review fairness and user profiling in machine learning.

2.1 User Profiling

User profiling has been a very important problem to solve in machine learning in the past years. User profiling is defined as trying to categorize users depending on their attributes like behavior, interest, and traits into specific groups. In order to solve this problem efficiently an important step before classifying the user is the data gathering process or how is it gathered. The data gathering process plays an important role in correctly classifying users into groups. A user profile usually consists of information that tries to represent a user depending on specific traits and properties. These properties may include settings, interests, behavior, and needs. Other than that user data must also reflect the user correctly so it can be accurate enough [14]. Nonetheless the amount of such data is also important, and this can vary strongly depending on the use case or the field of this problem.

There have been many approaches used for user profiling classification in the past years. These approaches depend on the derived features of the user profiles, for example like user interactions or the relationship to other users. Other features may also include features from texts [15]. Several machine learning models have been applied for this task in the last years as well as deep learning methods. For example in [1] the authors concentrate on classifying different users depending on several information such as text, images, and user relation. This was done by implementing a deep multimodal fusion model that can integrate different data types as input for better classification results.

Since nowadays user profiling applications depend more on social networks and e-commerce, it is also very important to have a better representation of the whole input data for the classification process. For example in the last years, e-commerce data were mainly used for recommender systems, and nowadays many machine learning approaches are depending more on graph data to have a better representation of the data. Using graph data in such classification problems, brings many advantages, for example using graph data we can better understand the relationship between the different users as well as their activities.

2.2 Graph Neural Networks

As discussed in the previous section, graphs are used in order to better model the relations between different users, as well as to model the complex relationships and their interactions. Similarly deep learning has gained more attention in the last years, and so it is the case with graph data. Graph neural networks **(GNNs)** have had a huge success in the last years because of their ability to analyze graph data [16, 17, 18]. Graph neural networks are a type of neural networks that try to learn node representation in graphs. The main concept behind GNN is its neural message passing framework in which messages of different vectors are exchanged between nodes in the graph and updated using neural networks. At each iteration of the message passing in a GNN, the embedding vector $h_u^{(k)}$ which corresponds to each node $u \in \mathcal{V}$ is updated depending on the information aggregated from the u's graph neighborhood $\mathcal{N}(v)$. The update mechanism of each node can be expressed as follows:

$$h_u^{(k+1)} = \text{UPDATE}^{(k)}\left(h_u^{(k)}, \text{AGGREGATE}^{(k)}(\{h_v^{(k)}, \forall v \in \mathcal{N}(u)\})\right) \tag{2.1}$$

where h_u is the node representation at each k iteration, u corresponds to each node $u \in \mathcal{V}$ in the graph, and N is the message passed to the neighbors of u. The AGGREGATE function normally is defined as taking the average or the summation of all nodes messages representation, while UPDATE is a learnable differentiable function which can be for example a Multi-layer perceptron (MLP). At the end of this process the node representations can be used for several tasks that use graph data like node classification, link prediction, and graph classification.

GNNs can be divided into two types, spatial-based GNNs [19, 20] and spectral-based GNNs [21, 22]. Spatial-based GNN approach imitates the convolution method used in Convolutional Neural Networks (CNN) that is famous for image classification and object recognition. In images the convolution is a predefined matrix (also called filter) that is applied to all of the image pixels and results in summing the neighboring pixels around the center pixel for each pixel. The same approach can be applied on graphs using Graph Convolutional Network (GCN) [23]. Which is interpreted the same way but instead of pixels we have nodes, and instead of the summation method we can aggregate the feature vectors of the neighboring node into the center node [23, 24].

Other improvements that have been made to this approach such as self-attention [20]. With the help of self-attention, specific nodes are given more weight or importance in the aggregation step. This was introduced in Graph Attention Network (GAT). Nonetheless, GCN has shown that it can suffer from scalability problems, since they require to store and pass node features and edge information every time which can be a problem in large graphs. Several approaches have been proposed to solve this problem [25], for example like GraphSage [25]. GraphSage mitigates this problem by using only a small sample of the nodes in the graph instead of the whole graph.

As discussed in the previous section, graph data are preferred in the last years for representing user profiles, along with GNNs for the classification. For example some approaches try to represent the data as heterogeneous graph learning, as they believe that user profiles may not only be reflected from users relationships but also from other entities (e.g. in e-commerce products that a user has purchased and their attributes) [6, 26]. Normally most of the approaches that were presented for heterogeneous graph learning tend to oversimplify the relations between different entities as a binary association between users using a vanilla GNN. A binary relation can mean for example in e-commerce applications that a user has either interacted with or without a specific product. This is done without giving importance to each specific relation. For example a liked product by the user can have less impact than a purchased product by the user. In order to tackle this problem, heterogeneous graph learning using attention was introduced [20] that can learn the representations of the different nodes and model multiple relations on the heterogeneous graph. Using transformer-like attention, they can learn different node representations and give more attention (e.g. weighting importance) to different attributes (e.g. liking a product vs. purchase of a product).

Other approaches try to improve the user profiling classification problem in GNNs with a pre-processing approach. For example in [7] they argue that the initial node representations in the graph largely affect the overall result and the model performance after training. This usually happens when the node representations are linearly combined embeddings of the node features, which does not account for the interactions among the feature embeddings. To solve this problem, they propose to represent the node features categorical vectors. They propose a new GCN model that takes such inputs for user profiling classification. This is accomplished by modeling local and global interactions into the learning of node representations. At the end both node interaction approaches are fused together using an aggregation approach to form the final node representation. Using such an approach, the model can capture the interactions among all node features in the graph. Furthermore to our knowledge, there has not been any work that included the pre-processing and training of several models in one framework for analyzing the results of user profile prediction for different GNNs models.

2.3 Fairness

Unfortunately GNNs may introduce some problems that can affect the quality of either its node representation or in node classification tasks. One problem is node discrimination towards sensitive attributes [27]. GNNs can aggravate the dependence of bias in node classification problems resulting in discrimination in training GNNs. This can happen as a consequence of the message passing structure and the aggregation of node information in GNNs layers. Several approaches have been developed to solve this problem in GNNs which can be divided into three categories: Pre-processing GNNs approaches, in-processing GNNs approaches, and post-processing GNNs approaches.

Pre-processing approaches try to pre-process graphs input data before inputting it to the model for training. These approaches can include for example, removal of disparate impact of attributes on the data [28], or weighting the labels that causes the dataset to be unfair after training [27]. Another method would be changing the labels of some data samples to remove the discrimination [27]. Nonetheless such unfair attributes can also be present in multimedia datasets like images, and deep learning approaches like Generative Adversarial Networks (GANs) [29] which can be used to generate a dataset that is fairer [30]. On the other hand, the in-processing approaches try to mitigate the effect of the unfair sensitive attributes during training state of the art machine learning models. Some examples of in-processing approaches can include using regularization techniques or using adversarial learning during the training phase of the model. An example for this approach would be FairGNN. Lastly post-processing approaches try to change the specific labels that cause the discrimination after the training of the model [31].

To our knowledge FairGNN is considered to be the state of the art GNN model that mitigates bias while training. The main advantage of its approach is the usage of adversary training, but it can also bring some disadvantages. Mainly such an approach can be sometimes challenging to re-implement in other GNN models, since it depends on the model structure, and every GNN model is built differently. This is why a new fairness approach that can be generalized for all GNN models has to be examined.

Wer will findet Wege wer nicht will findet Gründe.

"JIM ROHN"

3 Preliminaries

In this chapter we introduce the needed information that was used to build or framework. We first introduce the fairness metrics that were used to measure how fair the model predictions are. Our fairness metrics can be divided under two main concepts, namely the disparate impact and disparate mistreatment. Then we introduce the debiasing approaches that we choose for our framework. After that we introduce the state-of-the-art GNN models that we adopted for our framework and give information about their structure. Finally we introduce the datasets that were used in this work.

3.1 Fairness metrics

3.1.1 Notations

For the rest of the thesis, we will be defining our fairness metrics by taking into consideration some basic notations. First we define our user profiling prediction models as a transformation function of input data x to target label y as $f : x \to y$. Since we will be considering only binary classification in this work, we define our binary target label as $y \in \{0, 1\}$ and $\hat{y} \in \{0, 1\}$ as the model prediction for the user profile after training. We also denote for our sensitive attributes as $s \in \{0, 1\}$. Additionally we denote some classification properties as: **TP**, **FP**, **TN**, and **FN** as true positives, false positives, true negatives, false negatives respectively.

3.1.2 Disparate impact

We first introduce the concept of fairness in terms of disparate impact, it describes a type of covert, frequently unintentional discrimination that takes place when procedures or systems appear to treat people equally. Disparate impact focuses on instances where the model unfairly discriminates against particular groups, even when the model uses proxy attributes rather than the sensitive feature directly to produce predictions. Such case occurs when analysing GNNs models in training, where the sensitive attribute is not explicitly taken into account during classification and the user models are formed by combining data from neighbors.

The concept of disparate impact itself can also be depicted or used as a general fairness metric that is based on evaluating the proportion of individuals who receive a positive value depending on their sensitive attribute (e.g. age, and country). In another words disparate impact can be used to evaluate how fair the dataset is in regards to the chosen sensitive attributes. Disparate impact metric can be described as follows:

$$\frac{P\left(Y=1|D=unprivileged\right)}{P\left(Y=1|D=privileged\right)} \tag{3.1}$$

Similarly, we can evaluate the disparate impact value of analysed models through *statistical parity*, *equal opportunity*, and *overall accuracy equality* [13].

Statistical Parity which is also known as demographic parity which concentrates on group fairness rather than individual fairness. It defines fairness as having an equal positive probability for each group of predictions independent from the sensitive attribute [32].

$$SP = P(\hat{y}=1|s=0) = P(\hat{y}=1|s=1) \tag{3.2}$$

For our case, we calculate the statistical parity for our user profile predictions as the difference between the probability of positive model prediction given the negative sensitive attribute (s = 0) and the same probability given the positive sensitive attribute (s = 1).

$$\Delta_{SP} = |P(\hat{y}=1|s=0) - P(\hat{y}=1|s=1)| \tag{3.3}$$

Equal Opportunity assesses whether people of a specific group who should be eligible for an opportunity are equally likely to do so regardless of the defined sensitive attribute. In order to calculate equal opportunity metric, the probability of a person in a positive class has to be classified with the positive outcome, and should be valid for each group. This assures that the TP must be equal for all groups. The equal opportunity metric can be described in equation (3.4) as follows:

$$EOD = P(\hat{y} = 1|y = 1, s = 0) = P(\hat{y} = 1|y = 1, s = 1) \tag{3.4}$$

For our calculation we calculate the difference between both probabilities as follows:

$$\Delta_{EOD} = |P(\hat{y} = 1|y = 1, s = 0) - P(\hat{y} = 1|y = 1, s = 1)| \tag{3.5}$$

Overall Accuracy Equality is defined as the equal likelihood of a subject belonging to either the positive or negative class being allocated to its corresponding class. This can be for example represented in the following equation as:

$$\begin{aligned} OAE = P(\hat{y} = 0|y = 0, s = 0) + P(\hat{y} = 1|y = 1, s = 0) = \\ = P(\hat{y} = 0|y = 0, s = 1) + P(\hat{y} = 1|y = 1, s = 1) \end{aligned} \tag{3.6}$$

Similarly we calculate the overall accuracy equality for our framework as:

$$\begin{aligned} \Delta_{OAE} = |P(\hat{y} = 0|y = 0, s = 0) + P(\hat{y} = 1|y = 1, s = 0) - \\ -P(\hat{y} = 0|y = 0, s = 1) + P(\hat{y} = 1|y = 1, s = 1)| \end{aligned} \tag{3.7}$$

3.1.3 Disparate mistreatment

We also consider the concept of disparate mistreatment, which concentrates on user groups who have different sensitive attributes. This concept does not concentrate on the corrected predictions but on the misclassification of users that have different sensitive attributes values [33]. For this case, we calculate the treatment equality metric to account for this concept [3].

Treatment Equality demands that the classifier's error ratio be identical across all groups. In other words each group would have the same ratio of false negatives (FN) and the same ratio of false positives (FP). Treatment equality can be defined as follows:

$$TED = \frac{P(\hat{y} = 1|y = 0, s = 0)}{P(\hat{y} = 0|y = 1, s = 0)} = \frac{P(\hat{y} = 1|y = 0, s = 1)}{P(\hat{y} = 0|y = 1, s = 1)} \tag{3.8}$$

For our framework we calculate treatment equality as follows:

$$\Delta_{TED} = \left| \frac{P(\hat{y} = 1|y = 0, s = 0)}{P(\hat{y} = 0|y = 1, s = 0)} - \frac{P(\hat{y} = 1|y = 0, s = 1)}{P(\hat{y} = 0|y = 1, s = 1)} \right| \tag{3.9}$$

3.2 Debiasing approaches

As discussed in section 1.2, we need to find suitable approaches that can be applied for debiasing discrimination in GNNs and mainly in our framework. Since GNNs can aggravate bias while training on data with their message passing algorithm, we believe that by having a discrimination-free dataset the bias in training will be decreased in GNN. As a result, we present in this section three pre-processing debiasing approaches to mitigate bias in GNNs, namely sampling, reweighting, and disparate impact remover.

3.2.1 Sampling

The first chosen debiasing approach for our user study is sampling [34]. The sampling approach attempt to re-sample the dataset in a way so that the discrimination in the dataset is mitigated or removed. This is achieved by first dividing the data into 4 groups, and then we apply a uniform sampling approach to sample data from each group.

As discussed we first divide the dataset into four samples, mainly: Deprived community with positive class labels **(DP)**, Deprived community with negative class labels **(DN)**, Favored community with positive class labels **(FP)**, and Favored community with negative class labels **(FN)**. These sample groups are defined as follows:

In DP we sample all the data in our dataset that belongs to the positive class label (e.g. 1) and have an unfavorable sensitive attribute (e.g. 0).

$$\mathrm{DP} := \{X \in D | X(s) = - \wedge X(class) = +\} \tag{3.10}$$

Similarly in FP we sample the data group having the same positive class label, but this time while belonging to the favorable sensitive attribute group (e.g. 1).

$$\mathrm{FP} := \{X \in D | X(s) = + \wedge X(class) = +\} \tag{3.11}$$

In DN and FN we apply the same concept but with consideration to the negative class label (e.g. 0).

$$\begin{aligned} \text{DN} &:= \{X \in D | X(s) = - \land X(class) = -\} \\ \text{FN} &:= \{X \in D | X(s) = + \land X(class) = -\} \end{aligned} \tag{3.12}$$

After dividing the data into the four groups, we calculate for each class label and sensitive attribute the expected number of each item that has to be sampled from each of the data sample groups that we have created. This can be noted as follows:

$$\text{Let W (s, class)} = \frac{|\{X \in D|X(s) = s\}| \times |\{X \in D|X(class) = c\}|}{|D| \times |\{X \in D|X(class) = c \land X(s) = s\}|} \tag{3.13}$$

Finally we sample each of the groups separately, until the expected group size is achieved. Algorithm 1 shows a formal description of the whole approach.

Algorithm 1 Sampling

```
Input: (D, S, Class)
Output: Resampled D
 1: for $s \in \{w, b\}$ do
 2:     for $c \in \{-, +\}$ do
 3:         Let W (s, class) = $\frac{|\{X \in D|X(s)=s\}| \times |\{X \in D|X(class)=c\}|}{|D| \times |\{X \in D|X(class)=c \land X(s)=s\}|}$
 4:     end for
 5: end for
 6: Sample $W(b,+)$ $x$ $|DP|$ objects from $DP$
 7: Sample $W(w,+)$ $x$ $|FP|$ objects from $FP$
 8: Sample $W(b,-)$ $x$ $|DN|$ objects from $DN$
 9: Sample $W(w,-)$ $x$ $|FN|$ objects from $FN$
10: Let $\hat{D}$ be the output dataset of all samples generated in steps 6 to 9
11: return $\hat{D}$
```

3.2.2 Reweighting

In this approach we try to mitigate discrimination by reweighting [34], mainly by assigning weights to the dataset tuples. This is done by giving the unfavorable sensitive attribute higher weights than favorable sensitive attributes. For example, items having a sensitive attribute X(s) = 0 and prediction label X(class) = 1 will be getting higher weights than sensitive attribute X(s) = 0 and X(class) = 1. For this, we have to check the chosen dataset to find out which sensitive attribute label (e,g, 0 or 1) is being categorized as an unfavorable label. The next step will be calculating the required weight for each label. To know which weights to add for each item in the dataset, we need first to calculate the expected probability for a given sensitive attribute label and class label. In theory we assume that the dataset is unbiased and that each sensitive attribute and class label are statistically independent, this can be calculated as follows:

$$P(S = - \wedge Class = +) := \frac{|\{X \in D | X(s) = -\}|}{|D|} \times \frac{|\{X \in D | X(class) = +\}|}{|D|} \tag{3.14}$$

In reality there is bias in the dataset, and thus we need to calculate the observed probability of each sensitive attribute and class label as follows:

$$P(S = - \wedge Class = +) := \frac{|\{X \in D | X(s) = - \wedge X(class) = +\}|}{|D|} \tag{3.15}$$

If the observed probability is different than the expected probability for a specific sensitive attribute and a label class, mainly if the expected probability is higher than the observed probability this shows that we have bias toward the opposite class label.

To overcome the difference and the bias, we need to assign lower weights to tuples that are being favored. This is calculate as follows:

$$W(X) := \frac{P(S = X(s) \wedge Class = X(class))}{P(S = X(s) \wedge Class = X(class))} \tag{3.16}$$

After assigning the weights to each tuple we save the new data with the new weights.

3.2.3 Disparate impact remover

Our last debiasing approach for this work is disparate impact remover [35]. As the name suggests, this approach is a pre-processing algorithm that tries in a way to remove the disparate impact in the dataset. Disparate impact remover edits the data values to increase fairness between different sensitive attribute groups, while preserving the groups ranking between them, i.e. mitigating fairness in the dataset while not affecting the prediction result. To explain the disparate impact remover algorithm, we need to define some preliminaries. A dataset is defined as $D = (X, Y, C)$, where X is a protected attribute, Y is the unprotected attribute in the dataset, and C is the binary decision of the prediction (either 0 or 1). The goal of the disparate impact remover algorithm is to create a repaired version of D, i.e. $\hat{D} = (X, \hat{Y}, C)$. Where the attributes X in $\hat{D}$ are changed so that $\hat{D}$ does not contain disparate impact in terms of the protected values X.

We first define $Yx = Pr(y|X = x)$ be the marginal distribution on y condition on $X = x$. We denote $Fx : Yx \to Y[0,1]$ being the cumulative distribution function for values $y \in Yx$ and $F^{-1}x : [0,1]$ being the associated quantile function. As stated in [35] it is very important that the repaired version of the dataset $\hat{D}$ that it preserves the rank, if for any $y \in Yx$ and $x \in X$ its repaired counterpart has $Fx(y) = Fx(\hat{y})$.

We let A be a distribution such that $F_A^{-1}(u) =$ median, in which A is the distribution minimizing between Y_x and $C \sum x \in_X d(Y_x, C)$ over all distributions C, where $d(.,.)$ is the earth mover distance on R.

The repair algorithm first creates $\hat{Y}$, so that for all $y \in Y_x$ it correlates with $\hat{y} = F_A^{-1}(F_x(y))$. At the end the algorithm changes the dataset as $\hat{D} = (X, \hat{Y}, C)$, where only Y is changed, while the protected attribute and class are the same as in D.

3.3 Models

In this section we introduce the models used for our framework. As discussed in chapter 1, the framework is flexible and can contain more GNN models that can be added in the future. For the sake of this work, the first version of this framework contains three GNNs models, namely FairGNN, RHGN, and CatGCN.

3.3.1 FairGNN

The first model we use for our framework is FairGNN. FairGNN focuses on binary classification and was introduced to overcome the severe bias issues in GNNs predictions.

The main goal for FairGNN to is to learn a fair GNN node predictions given a graph with small labeled node set $G = (V, E, X), V_L \in V$, that corresponds

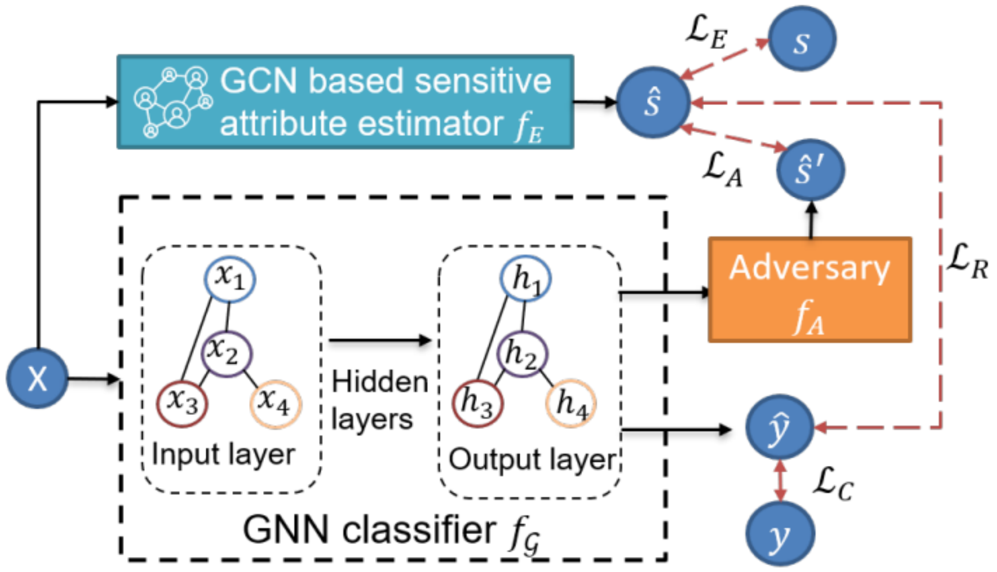


Figure 3.1: The overall architecture of the **FairGNN** model implemented in the framework [13].

to the labels in y, and set of nodes containing sensitive attribute values $V_S \in V$. This can be also noted mathematically as:

$$f(G, Y, S) \rightarrow \hat{Y} \tag{3.17}$$

where G is the input graph, Y is the labels, S represents the sensitive attributes, and $\hat{Y}$ is the fair GNN predictions. To achieve such a goal, the FairGNN framework is composed of three models, namely a normal GNN classifier (**FG**) which can be either a GCN model or a GAT model, a GCN sensitive attribute classifier (**FE**), and an adversary (**FA**). The whole framework structure is illustrated in Figure 3.1.

The first used model which is the GNN classifier (**FG**) takes a graph as an input data and tries to predict the binary node labels. In [13] they note that any vanilla GNN classifier can be used like GCN or GAT. As discussed in [13], they note that data with unlabeled sensitive attributes would result in a poor performance in the overall fairness. This is why the second model is used (**FE**), which is a sensitive attribute classifier. This model tries to predict the missing sensitive attributes values in the dataset. As well as estimating the sensitive attributes they note in [13] that this would help as well in the debiasing approach that was introduced. This paves the way for the third model i.e the adversary (**FA**). During the normal training process, the adversary model tries to predict the missing sensitive attributes in the dataset, while doing so the **FG** model tries to learn node representations that tries to fool the adversary predictions, i.e. it tries to make the **FA** unable to distinguish which sensitive group the node belongs to. The whole training process is trained using a min max game approach which can be noted as follows:

$$\min_{\theta_G \theta_E} \max_{\theta_A} \mathcal{L}_C + \mathcal{L}_E + \alpha \mathcal{L}_R - \beta \mathcal{L}_A \tag{3.18}$$

where θ_G, θ_E, and θ_A are the parameters to be trained for the GNN classifier, sensitive estimator classifier, and adversary respectively. α and β represented the pre-defined scalars by the user to control the covariance constraint and adversary debiasing.

Using this in-processing debias approach, FairGNN have shown great results in mitigating discrimination during GNN training while preserving the overall model accuracy.

3.3.2 RHGN

The second model that we use for our framework is Relation-aware Heterogeneous Graph Networks (RHGN). In comparison to FairGNN, RHGN tries to solve the problem of user profiling by concentrating on better representing the different user interaction types (e.g. user clicks on an item and user purchases an item). This is done by modeling the multiple user relations using a heterogeneous graph in comparison to the single relation graph that was used by previous approaches. For this RHGN depends on the heterogeneous graph structure to aggregate information from multiple sources, as well as using a transformer-like multi-relation attention mechanism. The attention mechanism helps in learning nodes according to their importance. The overall RHGN architecture is shown in Figure 3.2. As shown in Figure 3.2, RHGN consists of three parts, namely Neighborhood Message Passing, Multi-Relation Attention, and Target State Update. The first segment of the model, which is the Neighborhood Message Passing, which is responsible for collecting the node neighborhood messages, i.e. items that the node have interacted with. This is accomplished by calculating the message to pass in each layer, given that there exists a target node *t*,

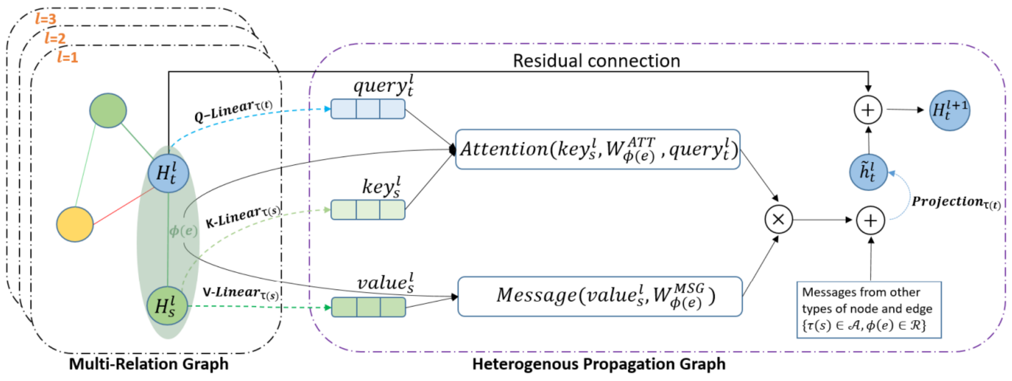


Figure 3.2: The overall architecture of the **RHGN** model implemented in the framework [6].

its neighbors s, and an edge relation $e = (s, t)$ that connects them together. This approach can be noted as:

$$\begin{aligned} M(s,e,t) &= \mathop{\Vert}_{i\in[1,h]} M-head^i(s,e,t) \\ M-head^i(s,e,t) &= F_M^i\left(H^l[s]\right) W_{\phi(e)}^{MSG} \end{aligned} \tag{3.19}$$

where the F^M is the i-th multi-head linear function, h is the number of heads, W is the matrix that shows the message into a relation-dependent space, and $\Vert$ is the concatenation operation.

The second part in the RHGN model is the Multi-Relation Attention, which is responsible for giving more attention to each message that is passed. Since not all messages have the same priority (e.g. which item the user has brought is more important than which advertisement the user has watched). This is adapted in [6] by projecting the source node and the target node into a Key vector and Query vector respectively and measuring the similarity between them. This can be noted as:

$$\begin{aligned} \alpha(s,e,t) &= softmax\left(\mathop{\Vert}_{i\in[1,h]} \alpha-head^i(s,e,t)\right) \\ \alpha-head^i(s,e,t) &= \left(K^i(s) W_{\phi(e)}^{ATT} Q^i(t)^T\right).\frac{1}{\sqrt{d}} \\ K^i(s) &= F_K^i(H^l[s]) \\ Q^i(t) &= F_Q^i(H^l[t]) \end{aligned} \tag{3.20}$$

where F_K^i is the multi-head linear function for the key vector, and F_Q^i is the multi-head linear function for the query vector, and W is the attention projection matrix. As discussed by adjusting the weight in the W matrix, the model can give more attention according to different meta relations in the graph.

Finally the last part in the RHGN is the Target State Update, which is responsible for updating the target node embedding after propagating the node messages and attentions as discussed in the previous steps. The defined update formula can be expressed as follows:

$$\begin{aligned} H^{l+t}[t] &= F_{map}\Big(\sigma(\hat{H}^l[t])\Big) + H^l[t] \\ \hat{H}[t] &= \bigoplus_{\forall s \in N(t)} (\alpha(s,e,t).M(s,e,t)) \end{aligned} \tag{3.21}$$

where F_{map} is the linear function that is responsible for mapping the messages back to the target feature distribution, and α represents the used activation function.

Although RHGN has shown great results in user profiling prediction using the presented approach, it was shown in [12] that it is not classified as a fair model as discussed in section 2.1.

3.3.3 CatGCN

Finally the last model used in our framework is CatGCN. CatGCN is adapted for graph learning on categorical node features. In [7], they argue that when input node features are categorical, they usually carry important information for prediction. Such information when ignored by other GNN architectures can greatly affect the prediction result at the end. As a result CatGCN tries to tackle this problem by introducing two explicit interaction modeling approaches for learning the node representation, namely local interaction and global interaction modeling. The overall architecture of CatGCN can be shown in Figure 3.3.

As discussed to be able to capture the interaction among all features two approaches are used, local interaction modeling and global interaction modeling. Local interaction modeling tries to capture the correlation between entities, for example gender and age (e.g. male with age between 20 and 25). To accomplish this, the multiplication operation is used by

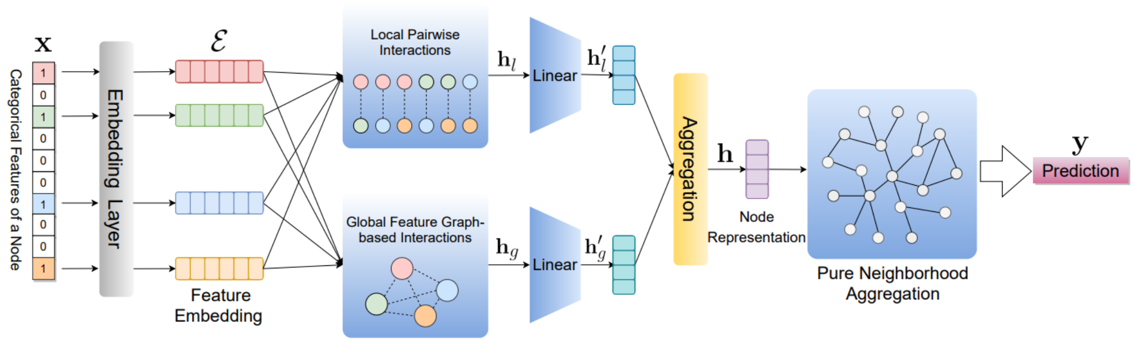


Figure 3.3: The overall architecture of the **CatGCN** model implemented in the framework [7].

different entities. Mathematically this can be formulated as the element-wise product of different feature embeddings. After that a representative operation can be used to aggregate the element-wise product on each pair of different feature embeddings. This is can be formulated as:

$$h_t = \sum_{i,j \in S - j > 1} e_i \odot e_j = \frac{1}{2}\left[\left(\sum_{i \in S} e_i\right)^2 - \sum_{i \in S} e_i^2\right] \tag{3.22}$$

where h denotes the initial node representation learned through local feature interaction modeling and e^2 denotes $e_i \odot e_j$.

The second approach used in CatGCN is global interaction modeling. The goal of this approach is to capture the different node information related to the predicted target. In [7], they argue that to consider these global interactions is like uncovering the signal along the spectrum or the details between the data to enhance the quality of the initial node representation. In [7], they accomplish the approach by first defining an artificial graph (P,E) that represents the nonzero features of the nodes, where P is the adjacency matrix and E represents the embedding of the node features. Formally the whole approach can be noted as follows:

$$\begin{aligned} \tilde{P} &= Q^{-\frac{1}{2}}(P + \rho O)Q^{-\frac{1}{2}} = \frac{P + \rho O}{|S| + \rho}, \\ h_g &= pool\left(\sigma(\tilde{P}EW)\right) \end{aligned} \tag{3.23}$$

where $\tilde{P}$ is the normalized adjacency matrix with a probability coefficient ρ that adjusts the frequency to be strengthened or not. Additionally Q represents the degree matrix of an artificial graph where O is the identity matrix, W is the weight matrix, σ is the chosen activation function (e.g. ReLU), and pool is a pooling function that aggregates the global interactions across the features.

3.4 Datasets

For the purpose of training the framework and applying the conducted fairness user study, we choose four datasets, mainly NBA, Pokec-z, Alibaba, and JD. We specifically choose these dataset for our framework evaluation so that we can compare how each model performs on each of the datasets in terms of prediction accuracy and fairness.

3.4.1 NBA

The NBA [1] dataset was the same dataset used in [13] for training the FairGNN model. This dataset is obtained from kaggle containing information about around 400 NBA basketball players from the 2016-2017 season. The dataset contains various information e.g. player height, player salary, player team, and age. The dataset was extended in [13] to obtain graphs that link the NBA players together. This was done by crawling the NBA players twitter so that they can obtain their relationships together. The classification task for this dataset is to classify the salary of the players, namely if the salary is over the median of salaries or not. Similarly we consider the player countries as a sensitive attribute for this dataset.

3.4.2 Pokec-z

Pokec [2] is the second dataset that was used in training the FairGNN model [13]. Pokec is a social network dataset that contains millions of users and is also popular in Slovakia. The dataset is similar to facebook and twitter, which contains information about users such as gender, age, and working field. The dataset also contains information about user relationship to other users which makes it a good social network dataset for GNNs training. The Pokec dataset is divided into two datasets, which are Pokec-z and Pokec-n. For our experiments in this work we will using only one

[1] https://www.kaggle.com/datasets/noahgift/social-power-nba
[2] https://snap.stanford.edu/data/soc-pokec.html

varaiation of the dataset which is the pokec-z dataset. Additionally since the dataset is very big, we only used a sample from the dataset that contains about 65,000 users. Additionally the classification task for this dataset is to predict the working field of the users. For the sensitive attribute, we consider the user region for this dataset.

3.4.3 Alibaba

Alibaba[3] is a public large-scale e-commerce user profiling dataset that contains about 350,000 users with information about their purchases and click rates. The dataset is based on the alibaba portal in china, and was used in both [6] and [7] for their model training and experiments. For our framework classification we consider the user gender as a prediction label and the user age as the sensitive attribute. Since we will be considering only binary classification and working with binary attributes in our experiments, we use the same pre-processing techniques used in [12] to binarize our chosen sensitive attribute (age) into two groups.

3.4.4 JD

JD[4] is the second dataset that was used in both [6] and [7] for model training and evaluation. JD is also a large-scale e-commerce platform based in China, which consists of users and items information which also has a click and purchase relationship between users. For our framework experiments we use the same version of the dataset that was used in [12] which is a lite version of the dataset that contains only 15% of the users and items interactions. We also use the same pre-processing approaches that was inspired and used in [12] such as the binarization of the sensitive attributes to be applicable for the binary prediction task. To keep the classification process consistent, we also choose for our classification task the user gender as a prediction label, and user age as the sensitive attribute.

[3] https://tianchi.aliyun.com/dataset/56
[4] https://github.com/guyulongcs/IJCAI2019$_{H}GAT$

Don't give up; I won't give up.

"THE GREATEST", SIA

4 System

In this section we introduce in detail our framework and the different components that are used. Figure 4.1 shows a visual overview of the overall framework structure. The framework is divided into four different components, namely **Pre-processing component**, **Debiasing component**, **Core component**, and finally the **Fairness calculations**.

4.1 Pre-processing component

The pre-processing component is the most important part of the framework, since its goal is to properly pre-process the user input data for all available models in the framework. This component also contains the debiasing component which is responsible for the debiasing approaches that are applied on the input if needed. A general overview of the structure of the pre-processing component can be seen in Figure 4.1.

As discussed in chapter 1 the framework takes as input a unified data format, namely either a NetworkX or a Neo4J data format. The idea of having a unified input data format for the framework helps the user to only give these two specific formats for all available models and for any chosen dataset.

The first step of the pre-processing component will be then to translate the given input data format to the format required for the selected models. The whole process of the decoding of the input data is also shown in Figure 4.2.

The process begins with checking if the user has inputted either a Neo4J or a NetworkX data format for the framework. For each of the data formats, we apply their specific decoding functions. The decoding function is mainly the conversion of the input file into a dataframe. For the network data format case, the framework supports several NetworkX formats (e.g. .graphml,

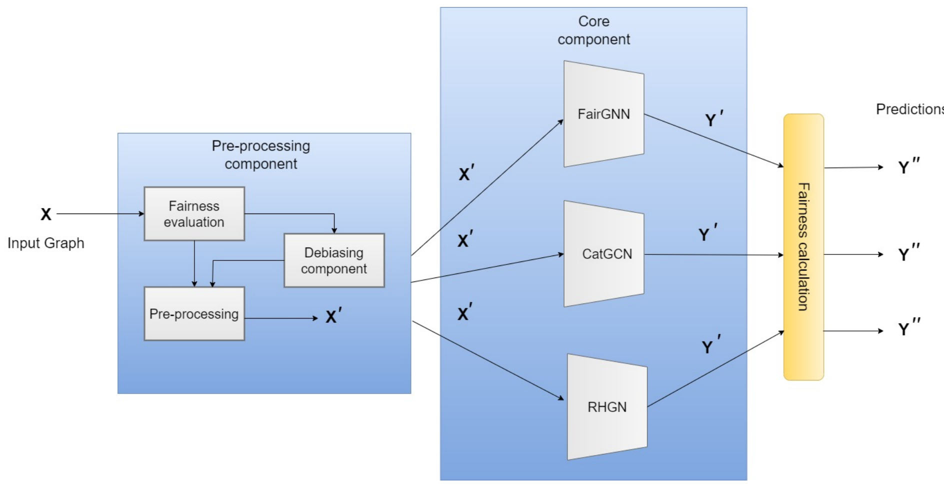


Figure 4.1: Overview of the novel framework overall structure and its different components.

.gexf, .gml, .leda, and .net). Several formats can be added later on in the next version of the framework. For these NetworkX formats the process of the conversion into a dataframe is easy, since there is a direct function available in the NetworkX library for this process.

For the Neo4J data format, we decode the dataframe manually since there is no direct function to decode the data. The Neo4J data format follows a standard schema, namely a json data schema, where each node or edge of the graph is represented as a json data format. We then decode each of the node json data as a new row in a dataframe, and each item in the properties section in the json data is decoded as a column for the row. To extract the edges, we apply the same procedure applied for the nodes, but we extract the relationship json data and map the start node id with the end node id as a row in our dataframe. A simple example of the json decoding for Neo4J data format is shown in Figure 4.3 and Figure 4.4.

As shown in Figure 4.2, when decoding a Neo4J data format we get already the edges needed, but since not all models need the edges for their training we only create edges for specific models (e.g. FairGNN). Regarding the NetworkX data format, when decoding this data format file into a dataframe we need to manually create the edges. Sometimes some datasets may come with the edges file (e.g. NBA and Pokec), but for any other dataset we create the edges using a specific function that we have created.

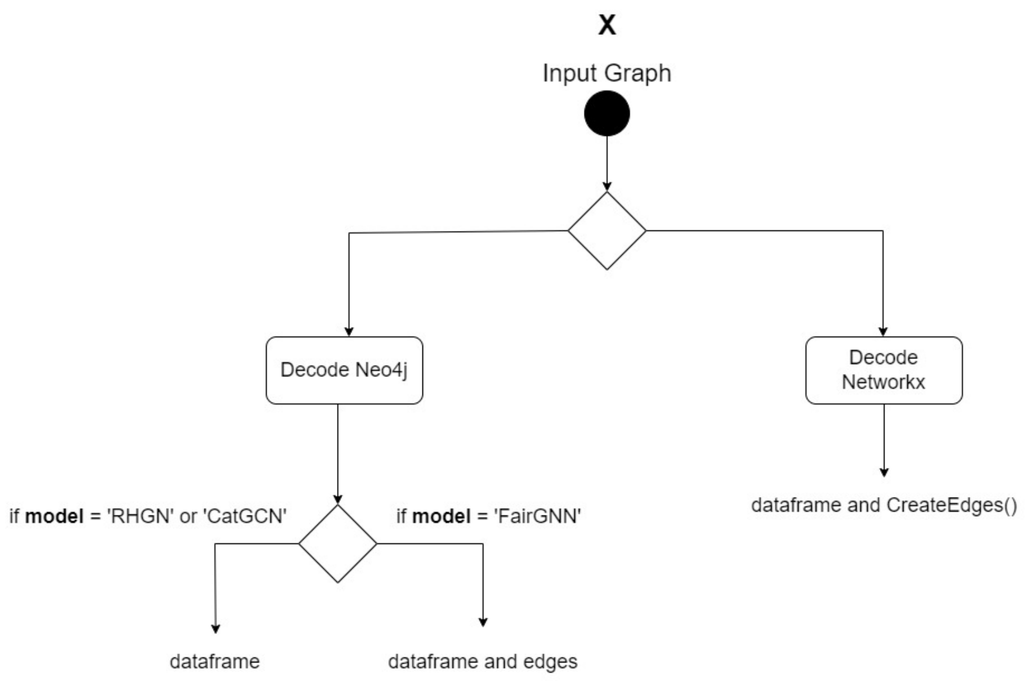
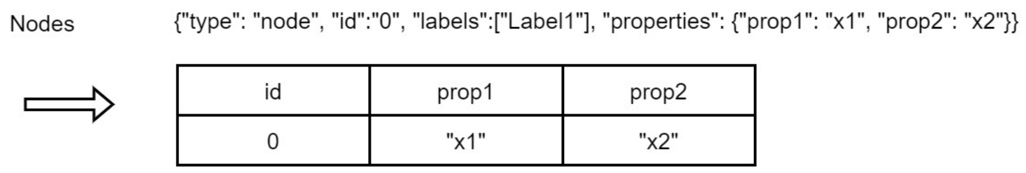


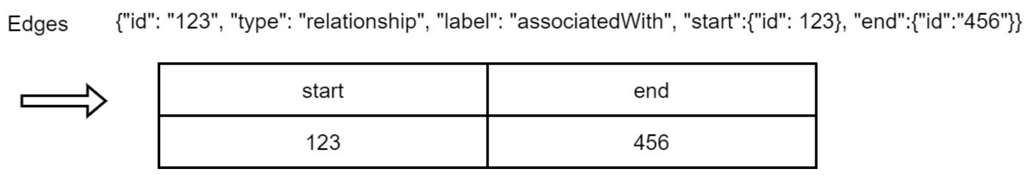
Figure 4.2: Visual overview of the data input decoding process in the pre-processing component.


Nodes

{"type": "node", "id":"0", "labels":["Label1"], "properties": {"prop1": "x1", "prop2": "x2"}}

id	prop1	prop2
0	"x1"	"x2"

Figure 4.3: Example of the **nodes** decoding using json Neo4J data format.


Edges

{"id": "123", "type": "relationship", "label": "associatedWith", "start":{"id": 123}, "end":{"id":"456"}}

start	end
123	456

Figure 4.4: Example of the **edges** decoding using json Neo4J data format.

4.1.1 Debiasing component

After decoding the input data to the specific data structure for each model, the next step is to evaluate the fairness in the dataset before preprocessing. This feature would need to be toggled manually from the user, since by default the framework does not evaluate the dataset fairness nor applies a debiasing approach on the dataset given as input. If the fairness feature is toggled in the framework the dataset would be evaluated as discussed in subsection 3.1.2 using the disparate impact metric. Depending on the evaluation result and a built-in threshold we debias the dataset using the following equation:

$$\hat{D} = \begin{cases} Debias(D), & \text{if } FE < 0.8 \\ D, & \text{otherwise} \end{cases} \tag{4.1}$$

where FE is the fairness evaluation and $Debias$ function is any of the chosen debias approaches available in the framework from the user. When the result of the evaluation metric is less than 80% we apply the debias mechanism. This threshold is the standard of the disparate impact violation, it also can be changed by the user if needed. As discussed if the dataset is considered not fair in regards with the fairness evaluation the debias approaches would be applied on the dataset. The user can choose from the different debias approaches discussed in section 3.2 (e.g. disparate impact remover, reweighting, and sampling) for the input dataset. The whole logic of the debaising component is presented in Algorithm 2, where $\mathcal{S}$ is the sampling debias approach, $\mathcal{R}$ is the reweighting debias approach, and $\mathcal{I}$ is the disparate impact remover debias approach.

4.1.2 FairGNN pre-processing

The FairGNN model uses a fairly simple pre-processing approach since FairGNN is a simple GCN or GAT model but with an additional in-processing debias approach. The FairGNN pre-processing is unified for all datasets that we have chosen in section 3.4.

After the data is decoded as discussed in the previous section, we get a dataframe and an edge dataframe file that maps which user is connected to the other user in the specified dataset. We then create an adjacency

Algorithm 2 Debiasing component.

```
Input: D
Output: $\hat{D}$ or D
 1: Calculate FE using disparate impact
 2: if FE < 0.8 then
 3:     if debias approach == Sample then $\mathcal{S}(D)$
 4:         return $\hat{D}$
 5:     end if
 6:     if debias approach == Reweighting then $\mathcal{R}(D)$
 7:         return $\hat{D}$
 8:     end if
 9:     if debias approach == Disparate Impact Remover then $\mathcal{I}(D)$
10:         return $\hat{D}$
11:     end if
12: else
13:     return D
14: end if
```

matrix using the created edges and user id in the original dataframe in a sparse matrix in coordinate format. After that we create a graph using the adjacency matrix created in the previous step. We use the DGL framework for the graph creation. This graph will be the main component for training the FairGNN model (either a GCN or a GAT model). We also split the dataset into different splits, namely a train, validation and test split, that will be also used in the training and testing the model later on.

4.1.3 RHGN pre-processing

In comparison to the FairGNN model the RHGN model pre-processing approach is a little bit more complex, since RHGN is mainly trained on a multi-relation graph. The pre-processing goal would be then preparing the dataframe created in the decoding step into a multi-relation graph. In addition to the multi-relation graph, we also need to extract the features needed to build the query, key, and value embedding vectors that will be needed in the training process of RHGN.

We divide the RHGN pre-processing approach into two cases. The first case is pre-processing datasets that are divided into several parts. An example for such a dataset would be the alibaba and JD dataset. These datasets are originally divided into three different datasets. For example in the alibaba dataset we have a user dataset which has all the user information (e.g. user id, age, and city), a item dataset which summarizes all the available items in the alibaba website (e.g. item id, brand name, and price), and finally a click dataset which outlines all the user and items interaction. For this case we use the original pre-processing procedure used in the original RHGN paper. The pre-processing approach starts with binarizing the sensitive attributes and labels in the dataset, since we are dealing with a binary classification problem. Then we clean all three datasets, by removing for example unneeded columns and dropping duplicates and null rows. After this we choose the three needed features, for both alibaba and JD dataset, we choose the same three features that the original paper chose which are at the end specific columns in the dataset. We then use a fasttext NLP model to extract the features from these three columns [40]. Finally we create a heterogeneous graph using the DGL framework by defining a user and an item node. The item nodes consist of the features that we extracted, and the user nodes are the rest of the dataframe columns.

The second case is pre-processing datasets that are originally not divided into several datasets. Such examples include the NBA and pokec datasets. Since in the original paper implementation they did not use such a dataset, We needed to implement a similar pre-processing approach to achieve the same outcomes as in the first case. In our implementation for this approach we followed the same steps done in the original paper pre-processing. We begin by cleaning and a binarization step for both the sensitive attributes and the labels. Since we only have one dataset to work with, it will be harder to get features for the model. To solve this problem, we select the three most important features for such datasets (e.g. in NBA: age, country and match played). We then follow the same features extraction approach by using the fasttext model, and create the heterogeneous graph using these features.

4.1.4 CatGCN pre-processing

CatGCN pre-processing approach is somehow similar to RHGN, since we implement pre-processing for two different cases as well. The first case is similar to RHGN, namely datasets that are originally divided into several datasets, and the second case is datasets that are not.

The output of the CatGCN pre-processing approach is mainly a user edge index mapping between users to build the main graph, a user field which is the index mapping of the users and items, and a target index mapping. In the first case for example the user field is the mapping of the user and the purchased items in the alibaba site.For the second case we choose a specific attribute to map it as an item (e.g. NBA: we choose user id and match played). Creating the user edge in the first case is simple, since we can map the different users regarding the item they bought (e.g. in alibaba and JD dataset). For both alibaba and JD dataset we use the same pre-processing approach that was used in the original paper to create the user edges. For the second case, namely for both the NBA and pokec dataset, we use the user edges dataframe that comes with both dataset as our pre-processed user edges for the CatGCN model.

The target index mapping is the mapping of the user id and the chosen prediction label for a specific dataset. For creating the target index mapping, we create a new dataframe that maps each user with the user id to its specific prediction label. An example would be a user having an id of 1 is mapped to its prediction label (e.g. gender: Female).

4.2 Core component

After the inputted dataset is correctly preprocessed, the next step would be to train the data on the model or models that the user has chosen in the framework. The user has the possibility to choose between different options for the training in the framework. The first option would be training a specific model from the models available in the framework. In this option the user needs to specify which model he wants to use (e.g. FairGNN, RHGN, CatGCN). The second option would be training two models sequentially. In this option the user also needs to specify which two models he wants to use (e.g. FairGNN and RHGN together). Finally the last option would be to train all the models available in the framework with the preprocessed data. The last option is the one that is activated by default in the framework and training of all models will happen sequentially.

To be able to correctly train the chosen models sequentially without any problems, an algorithm for scheduling the training of several models was also needed. Normally for all models in the framework, the final objective would be to train a preprocessed dataset X' until convergence and output the model predictions Y'. For each model in the framework, we used the same training algorithm that was used in their original papers. For all models in the framework this can be noted as follows:

$$f_{\mathcal{M}}(\mathcal{F}) \rightarrow \mathcal{Y}' \tag{4.2}$$

where $f_{\mathcal{M}}$ is the main training function of all models, $\mathcal{F}$ is the list of models that the user has chosen for training (note: the list may also contain one model), and $\mathcal{Y}'$ is the model predictions on the chosen dataset.

The list of $\mathcal{F}$ is noted as follows:

$$\mathcal{F} = \{f_{\mathcal{F}}, f_{\mathcal{R}}, f_{\mathcal{C}}\} \tag{4.3}$$

where $f_{\mathcal{F}}$ is the FairGNN training algorithm, $f_{\mathcal{R}}$ is the RHGN training algorithm, and $f_{\mathcal{C}}$ is the CatGCN training algorithm.

The training algorithm of the core component of our framework is presented in Algorithm 3. We first check how many models does the user want to train by checking the length of the list $\mathcal{F}$. We then, depending on the number of models in the list, apply specific conditions. If the user chooses only one model to be used for training we apply the specific training algorithm depending on the model. Similarly, is the case with having two models in the list $\mathcal{F}$. Else by default if the user did not specify any specific models to be trained we train all the models in the framework. Keep in mind that we only preprocess the given data according to the chosen model by the user. Likewise if the user did not specify any model we preprocess the data for all models to be ready for training.

4.3 Fairness calculations

After the chosen models by the user are trained, we can apply our defined fairness metrics calculations that were described in section 3.1 on the

Algorithm 3 Training Algorithm of the Core component.

```
Input: $\mathcal{X}'$
Output: $\mathcal{Y}'$
 1: Check models in $\mathcal{F}$
 2: if model count in $\mathcal{F}$ == 1 then
 3:     if FairGNN $\in \mathcal{F}$ then
 4:         Train FairGNN using $f_{\mathcal{F}}$
 5:     end if
 6:     if RHGN $\in \mathcal{F}$ then
 7:         Train RHGN using $f_{\mathcal{R}}$
 8:     end if
 9:     if CatGCN $\in \mathcal{F}$ then
10:         Train CatGCN using $f_{\mathcal{C}}$
11:     end if
12: end if
13: if model count in $\mathcal{F}$ == 2 then
14:     if FairGNN and RHGN $\in \mathcal{F}$ then
15:         Train FairGNN using $f_{\mathcal{F}}$ and RHGN using $f_{\mathcal{R}}$
16:     end if
17:     if FairGNN and CatGCN $\in \mathcal{F}$ then
18:         Train FairGNN using $f_{\mathcal{F}}$ and CatGCN using $f_{\mathcal{C}}$
19:     end if
20:     if RHGN and CatGCN $\in \mathcal{F}$ then
21:         Train RHGN using $f_{\mathcal{R}}$ and CatGCN using $f_{\mathcal{C}}$
22:     end if
23: else
24:     Train all models $f_{\mathcal{F}}$, $f_{\mathcal{R}}$, $f_{\mathcal{C}}$
25: end if
26: return $\mathcal{Y}'$
```

models predictions. The goal of this small part of the framework is to check if the chosen trained model is producing fair predictions. As discussed in section 2.3, GNN can aggravate discrimination the model predictions because of the message passing algorithm. This is why having a fairness calculation step in such a framework would help the user to recognize if his chosen trained model is producing fair predictions or not. Additionally the fairness calculation step can be also disabled by the user if needed, but by default the framework will calculate these metrics. Algorithm 4 shows how such structure is implemented in our framework.

Algorithm 4 Fairness calculations.

```
Input: $\mathcal{Y}'$
Output: $\mathcal{Y}''$
1: Calculate Statistical parity difference → $SPD(Y')$
2: Calculate Equal opportunity → $EOD(Y')$
3: Calculate Overall accuracy equality → $OAE(Y')$
4: Calculate Treatment equality → $TED(Y')$
5: Let Y” be the output set of both the predictions metrics and fairness metrics
   calculation.
6: return $\mathcal{Y}''$
```

The algorithm in Algorithm 4 applies for all models in our framework, where $\mathcal{Y}'$ is the model predictions and $\mathcal{Y}''$ is the output list of the model, where it contains both the model prediction metrics (e.g. accuracy, and F1 metric) and fairness metrics results. We also assume that the user want to calculate how fair the model predictions are, otherwise if the user disabled this step the model predictions are only given as output.

If you can dream it, you can achieve it.

ZIG ZIGLAR

5 Evaluations and results

In this section we present and discuss the experiments done for our framework and the fairness user study. We first present the fairness evaluation done before the training for all datasets, then we first introduce the key statistics of our chosen datasets for the experiments. Next we present the hyperparameters chosen for each model during the training process for each dataset. Finally we introduce and discuss the results of our experiments done using our framework.

5.1 Experimental settings

5.1.1 Dataset statistics

As discussed, for testing our framework we conduct a user study to examine the proposed debiasing approaches in comparison to the state of the art GNN fairness model FairGNN. Since FairGNN has two model variations proposed in its original paper, namely a GCN model and a GAT model, we use for our experiments the GCN model so that we can have a fair comparison with all of the proposed models in our experiments. We train all three models using our framework sequentially each time with a different dataset. We trained the models on the datasets proposed in section 3.4. We use the networkx data format for all of the datasets as our input data format for the framework. We first calculated how fair is the chosen dataset using the disparate impact fairness calculation. The results for all dataset is shown in Table 5.1. For the sake of the user study and experiments we ignore that the results are not less than 80% as defined and proceed with the debiasing approaches anyway.

To have a fair comparison between all datasets, we sample both Alibaba and JD with a fraction of 0.05 and 0.04 respectively, since both datasets

Dataset	Fairness
NBA	0.92
Pokec	1.008
Alibaba	1.57
JD	1.15

Table 5.1: Comparison of the different datasets fairness evaluation.

Dataset	Alibaba	JD	NBA	Pokec
Number of nodes	451,977	458,953	403	67,797
Number of node attribute	9	14	39	59
Number of edges	91,539	47,344	16,570	882,765
Sensitive Attribute	age	age	country	region
Prediction Label	gender	gender	salary	working in field

Table 5.2: Statistics of the datasets used in the experiments.

were very large in comparison to the other datasets. Furthermore, for all datasets we eliminate nodes without any links with other nodes. We use the same sensitive attributes and prediction label as discussed for each dataset in section 3.4. The key statistics of all datasets are available in Table 5.2. For each dataset, we randomly select 80% of the nodes to form the training set, 10% for the validation set, and 10% for the test set. Furthermore we apply the debiasing approaches only for RHGN and CatGCN, since FairGNN has its debiasing approach already implemented in it and we want to compare it to the other two models.

5.1.2 Hyperparamater selection

For all models in our framework, we tune the hyperparameters on the training set of each of the chosen dataset to obtain the best parameters for each model. For FairGNN we have three main hyperparameters to tune, which are the number of hidden layers in the model, alpha, and beta. α controls the influence of the adversary to the GNN classifier, while β controls the contribution of the covariance constraint to ensure fairness.For RHGN we tune the number of hidden layers. Finally for CatGCN we tune the weight decay, the dropout percentage, the diagonal probe coefficient, the field feature coefficient of the global interaction modeling (graph refining), field

Dataset	sens number	label number	hidden layers	α	β	learning rate
NBA	50	100	128	10	1	0.001
Pokec	200	100	128	100	1	0.001
Alibaba	500	200	256	200	150	0.0001
JD	50	10	64	50	5	0.0001

Table 5.3: Hyperparameters used for **FairGNN** on different datasets.

Dataset	hidden layers	clip	learning rate
NBA	64	1	0.01
Pokec	64	1	0.01
Alibaba	32	2	0.01
JD	32	2	0.01

Table 5.4: Hyperparameters used for **RHGN** on different datasets.

feature aggregation pooling approach (e.g. mean, sum), the hidden units for global interaction modeling (grn units), and the user feature computation. The different choices of the parameters are all described for the user in the framework description for each argument. Table 5.3, Table 5.4, and Table 5.5 show all the used hyperparameters for each dataset for FairGNN, RHGN, and CatGCN respectively.

5.1.3 Evaluation metrics

For our evaluation metrics, we used primarily the accuracy (**ACC**) and **F1** score as our metrics to measure how good our models perform on the binary classification task on the given datasets.

Regarding our fairness evaluation, we used the metrics proposed in section 3.1, namely statistical parity Δ_{SP}, equal opportunity Δ_{EO}, overall accuracy equality Δ_{OAE}, and treatment equality Δ_{TED} to show the discrimination level after training. The smaller the fairness metrics are (close to 0) the more fair the classifiers are. We had to add all of these fairness metrics

Dataset	weight decay	dropout	diag probe	graph refining	aggregation pooling	grn units	bi-interaction
NBA	1e-5	0.5	55	agc	mean	64	nfm
Pokec	1e-5	0.3	30	agc	mean	64	nfm
Alibaba	1e-5	0.1	1	agc	mean	64	nfm
JD	1e-2	0.1	39	agc	mean	64	nfm

Table 5.5: Hyperparameters used for **CatGCN** on different datasets.

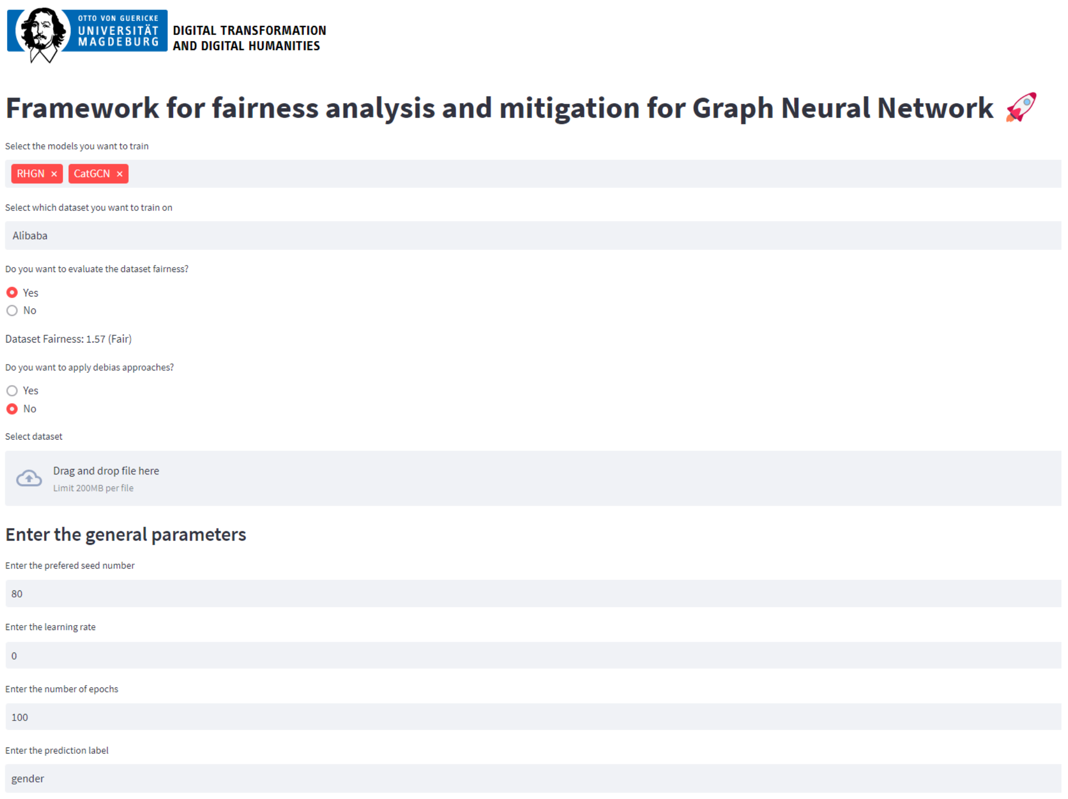
in our proposed models since in the original implementations didn't have these metrics implemented. Regarding the user study, we train each model 5 times, the mean and standard deviation for all the models on each dataset is the calculated.

5.2 Initial experiment

We first begin with a simple experiment to test the effectiveness of our framework. The goal of this experiment is to use the framework to train two models sequentially on a dataset and calculate the prediction accuracy and fairness metrics at the end. We choose RHGN and CatGCN as our models, and alibaba as our training dataset. To make the user experience for our framework seamless as possible, we built a simple web app using streamlit [36] which is an open-source library to build web apps. This also helps the users to input the required hyperparameters of their choice easily.

Upon loading the web app the user is required to choose which model to train (multiple models can be chosen), then which dataset to train on. The user then has the possibility to apply debiasing approaches, when this option is toggled the initial dataset fairness will be calculated before the training begins. For our initial experiment we will not use debiasing. Finally the user needs to select the dataset path. Depending on the model chosen, specific hyperparameters options will be available for the user to adjust, each hyperparameter will be grouped depending on their specific model. hyperparameters like seed number, learning rate, and epochs number are grouped under the general parameters options. All of these hyperparameters chosen will be then mapped as arguments for the framework main function. This web app is the initial version which means additional improvements and bug fixes will be added in later versions. Figure 5.1 gives an example of the framework web app.

Upon adjusting all the needed hyperparameters the user can begin training using the "Begin experiment" button. After the training is finished the training results as well as the fairness metrics for all chosen models will be displayed as a table as shown in Figure 5.2. At the end all of the experiment logs and metadata are stored on Neptune [37], which is a web app to store machine learning models training metadata and logs. The user has the ability to also view these logs after the training in the given folder path in


OTTO VON GUERICKE UNIVERSITÄT MAGDEBURG
DIGITAL TRANSFORMATION AND DIGITAL HUMANITIES

Framework for fairness analysis and mitigation for Graph Neural Network 🚀

Select the models you want to train

RHGN × CatGCN ×

Select which dataset you want to train on

Alibaba

Do you want to evaluate the dataset fairness?

[✓ Yes]
No

Dataset Fairness: 1.57 (Fair)

Do you want to apply debias approaches?

Yes
[✓ No]

Select dataset

Drag and drop file here
Limit 200MB per file

Enter the general parameters

Enter the prefered seed number

80

Enter the learning rate

0

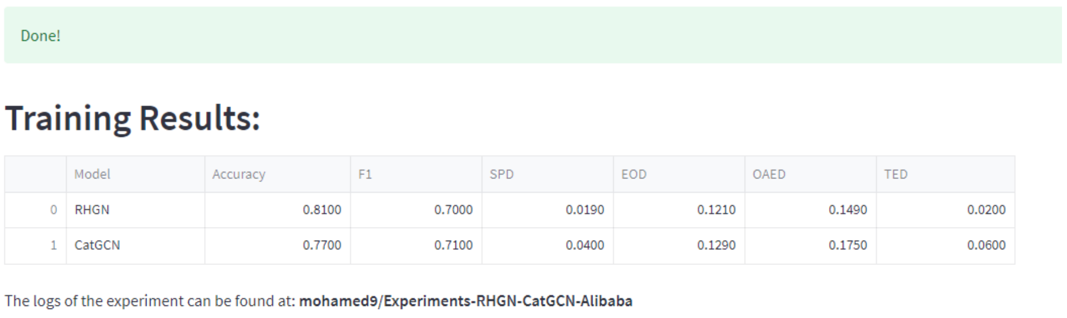
Enter the number of epochs

100

Enter the prediction label

gender

Figure 5.1: Example of the web app UI built for the framework.


Done!

Training Results:

	Model	Accuracy	F1	SPD	EOD	OAED	TED
0	RHGN	0.8100	0.7000	0.0190	0.1210	0.1490	0.0200
1	CatGCN	0.7700	0.7100	0.0400	0.1290	0.1750	0.0600

The logs of the experiment can be found at: **mohamed9/Experiments-RHGN-CatGCN-Alibaba**

Figure 5.2: Example of the framework results using the built web app UI.

Neptune as shown in Figure 5.2. This of course require the users to input their Neptune project name and security token in the framework fields so that the framework can log the results there. With this initial experiment we can observe that the framework is able to pre-process the different inputs to the chosen model by the user, thus making the framework able to standardize the different inputs.

5.3 NBA

We begin our fairness user study by training our models on the NBA dataset. From this experiment onwards, we will be training first all models normally without any debias approaches and then using the debias approaches to compare them to FairGNN. As discussed previously all of the model training (also using the debias approaches) are conducted 5 times. We use some abbreviations for the debiasing approaches for simplicity: Sample (S), Reweighting (R), and Disparate impact remover (D). This experiment will also be interesting for both RHGN and CatGCN, since in the original paper they didn't were not trained on NBA before. Table 5.6 shows all the models results for the NBA datasets.

From the results in Table 5.6 we can observe the following:

- Training both RHGN and CatGCN without debiasing on the NBA dataset, gives us somehow a comparable result in comparison to FairGNN performance on the same dataset. A reason for this can be that FairGNN uses the adversary to mitigate bias using the min max game approach, which helps in achieving a better prediction

	Performance		**Fairness**			
Models	Acc	F1	Δ_{SPD}	Δ_{EOD}	Δ_{OAED}	Δ_{TED}
FairGNN	**0.713 ± 0.018**	**0.744 ± 0.008**	**0.059 ± 0.022**	0.093 ± 0.073	0.183 ± 0.07	**0.190 ± 0.150**
RHGN	0.61 ± 0.05	0.57 ± 0.05	0.152 ± 0.09	0.197 ± 0.152	0.308 ± 0.161	0.375 ± 0.225
RHGN (S)	0.608 ± 0.059	0.549 ± 0.02	0.112 ± 0.074	0.160 ± 0.108	0.290 ± 0.055	0.36 ± 0.182
RHGN (R)	0.61 ± 0.05	0.57 ± 0.05	0.152 ± 0.09	0.197 ± 0.152	0.308 ± 0.161	0.375 ± 0.225
RHGN (D)	0.585 ± 0.06	0.488 ± 0.05	0.202 ± 0.007	0.546 ± 0.172	0.86 ± 0.386	0.447 ± 0.417
CatGCN	0.65 ± 0.12	0.656 ± 0.124	0.151 ± 0.130	**0.05 ± 0.037**	0.197 ± 0.131	0.260 ± 0.198
CatGCN (S)	0.605 ± 0.06	0.582 ± 0.08	0.131 ± 0.076	0.171 ± 0.034	**0.09 ± 0.050**	0.223 ± 0.259
CatGCN (R)	0.675 ± 0.08	0.622 ± 0.141	0.111 ± 0.116	0.069 ± 0.09	0.160 ± 0.136	0.442 ± 0.255
CatGCN (D)	0.51 ± 0.111	0.45 ± 0.117	0.118 ± 0.007	0.119 ± 0.132	0.311 ± 0.278	0.375 ± 0.05

Table 5.6: Comparison of the results using different models from the framework on the **NBA** dataset.

accuracy for the FairGNN model while keeping the fairness metrics fairly low.

- In regards to using the debiasing approach for the RHGN model, we can observe that using all debiasing approaches give us a very comparable accuracy result in comparison to the original RHGN model. Other than that, we can also see from the debiasing approaches results that the **sampling** approach gives us the best results in all metrics.
- Training the NBA dataset using the CatGCN model also brings comparable results in terms of fairness in comparison to RHGN. We observe for example that when using the sample approach we get some decrease in some fairness metrics (e.g. OAED). A reason for this can be that using the sample approach it generates new data to overcome discrimination in the dataset, this can be a key factor while training with small sized data like NBA to achieve a better prediction accuracy, and minimum discrimination in the predictions as well. Nonetheless, most of the applied debiasing approaches on the CatGCN model gives us also comparable results in terms of accuracy when compared to the original CatGCN model without any debiasing approaches.
- Lastly we can observe from the results, that FairGNN has still the best fairness metrics using the adversary debiasing approach. In comparison to the other approaches applied on both RHGN and CatGCN we can observe some decrease in the fairness metrics for some specific metrics but not all of them, but while having a decrease

	Performance		**Fairness**			
Models	Acc	F1	Δ_{SPD}	Δ_{EOD}	Δ_{OAED}	Δ_{TED}
FairGNN	0.646 ± 0.011	**0.652 ± 0.022**	0.034 ± 0.024	0.045 ± 0.024	0.053 ± 0.010	0.088 ± 0.052
RHGN	0.947 ± 0.002	0.487 ± 0.001	**0.002 ± 0.002**	0.002 ± 0.002	0.002 ± 0.002	**0.002 ± 0.001**
RHGN (S)	0.947 ± 0.001	0.487 ± 0.001	0.004 ± 0.001	**0.002 ± 0.001**	0.002 ± 0.022	0.003 ± 0.001
RHGN (R)	0.944 ± 0.001	0.485 ± 0.002	0.004 ± 0.001	0.010 ± 0.008	0.004 ± 0.005	0.004 ± 0.004
RHGN (D)	**0.948 ± 0.005**	0.486 ± 0.125	0.003 ± 0.012	0.008 ± 0.004	0.020 ± 0.006	0.022 ± 0.003
CatGCN	0.660 ± 0.005	0.50 ± 0.012	0.014 ± 0.004	0.383 ± 0.128	0.382 ± 0.200	0.006 ± 0.008
CatGCN (S)	0.690 ± 0.011	0.56 ± 0.010	0.023 ± 0.005	0.019 ± 0.010	0.090 ± 0.004	0.122 ± 0.040
CatGCN (R)	0.727 ± 0.005	0.494 ± 0.004	0.016 ± 0.009	0.007 ± 0.001	0.007 ± 0.004	0.037 ± 0.008
CatGCN (D)	0.510 ± 0.010	0.492 ± 0.020	0.002 ± 0.001	0.007 ± 0.008	**0.002 ± 0.001**	0.050 ± 0.010

Table 5.7: Comparison of the results using different models from the framework on the **Pokec** dataset.

in the model accuracy (e.g. OAED using the sample approach in CatGCN).

Overall this experiment has shown us that when training on the NBA dataset FairGNN still gives us the best model accuracy and lowest fairness metrics compared to the other models.

5.4 Pokec

The next experiment done is on the pokec dataset. This dataset was also used for the training of FairGNN in its original paper. Similar to the previous experiment (NBA), we will be comparing the overall prediction results and fairness metrics using the different debiasing approaches to the FairGNN results. Furthermore, we first train the original models without any debiasing approaches for 5 runs, and then train them again using the different debiasing approaches. This experiment will be the second experiment for both RHGN and CatGCN on new data other than the datasets presented in both of the models original papers. Table 5.7 displays the experimental results for all models and all debiasing approaches.

After this experiment and upon the presented results, we can conclude the following:

- The RHGN model does perform well on the pokec dataset, this can be seen for example upon observing the model accuracy and F1 score. We can see that there is a huge difference upon these two metrics, which can conclude that the model may have overfitted on the data. We tried adjusting the different hyperparameters to overcome this problem, but we tend to get the same result. These results also strongly affect the fairness metrics results. For example we can see from Table 5.7 that RHGN has very low fairness metrics, but such fairness metrics results would be then invalid in regards to the accuracy and F1 score.

- On the other hand, CatGCN performs much better in comparison to RHGN on the pokec dataset without using any debiasing approaches. This for example can be shown by observing the model prediction accuracy and F1 metric. The CatGCN accuracy results is very comparable to FairGNN as well.
- In terms of debiasing approaches, CatGCN shows very comparable results to FairGNN having also lower discrimination in the predictions in some metrics. This for example can be seen when using the sample approach (e.g. SPD and EOD metrics). Furthermore the reweighting approach shows fairly similar results in comparison to the sample approach while having a better prediction accuracy. At the end this shows in general that the debiasing approach does not target all the fairness metrics but some of them. The disparate impact violation approach gives us the worst prediction accuracy in comparison to FairGNN, but still gives us fairly low fairness metrics as well.

To conclude, from the observed results of this experiment, we can see how FairGNN and CatGCN have fairly similar results in regards to prediction accuracy and fairness metrics when trained on the pokec-z dataset, with CatGCN having better fairness results when applying the reweighting approach.

5.5 Alibaba

The third experiment that was done, is on the Alibaba dataset. The Alibaba dataset was used in both the RHGN and CatGCN original paper. Furthermore they were also used in [12], where it was first noted that these models were not fair during user profiling classification. Again like the previous experiment, we train all models without any debiasing approaches for 5

	Performance		**Fairness**			
Models	Acc	F1	Δ_{SPD}	Δ_{EOD}	Δ_{OAED}	Δ_{TED}
FairGNN	0.74 ± 0.004	0.43 ± 0.001	0.017 ± 0.015	**0.002 ± 0.008**	**0.001 ± 0.019**	**0.008 ± 0.007**
RHGN	0.802 ± 0.002	0.709 ± 0.003	0.021 ± 0.007	0.116 ± 0.004	0.144 ± 0.003	0.024 ± 0.004
RHGN (S)	0.800 ± 0.003	0.703 ± 0.003	0.019 ± 0.005	0.028 ± 0.003	0.148 ± 0.022	0.024 ± 0.005
RHGN (R)	**0.825 ± 0.002**	0.708 ± 0.004	0.023 ± 0.003	0.115 ± 0.008	0.144 ± 0.001	0.026 ± 0.006
RHGN (D)	0.719 ± 0.003	0.478 ± 0.002	**0.004 ± 0.002**	0.072 ± 0.004	0.101 ± 0.006	0.028 ± 0.003
CatGCN	0.808 ± 0.025	**0.717 ± 0.003**	0.030 ± 0.014	0.040 ± 0.018	0.030 ± 0.016	0.051 ± 0.032
CatGCN (S)	0.789 ± 0.014	0.717 ± 0.006	0.027 ± 0.007	0.047 ± 0.010	0.039 ± 0.009	0.054 ± 0.024
CatGCN (R)	0.75 ± 0.024	0.690 ± 0.004	0.046 ± 0.022	0.046 ± 0.012	0.128 ± 0.007	0.040 ± 0.020
CatGCN (D)	0.72 ± 0.05	0.660 ± 0.002	0.055 ± 0.043	0.051 ± 0.008	0.167 ± 0.004	0.092 ± 0.010

Table 5.8: Comparison of the results using different models from the framework on the **Alibaba** dataset.

runs, and do the same using each debias approach. This experiment will be the first test for FairGNN on new data as well, since it will be interesting to see how FairGNN behaves on different data other than the ones presented in the original paper. Table 5.8 shows the experiment results using the different models and debiasing approaches.

After viewing the models results in Table 5.8, we can conclude the following:

- We can first see that FairGNN prediction results are very similar compared to both RHGN and CatGCN (acc: 74%). But we can also observe that the F1 score is not the best. This can be caused by several reasons, one reason could be that the model is having problems classifying only one specific class and ignoring the other. This also can have an effect on the fairness metrics, since we can observe from the results that FairGNN has low fairness metrics in comparison to the other models. Those fairness metrics results can be interpreted as invalid for such cases. In the end, this is a preliminary experiment, and we believe there is room for improvement for FairGNN on the Alibaba dataset.

- Then we can see from the results that the original RHGN model (without debiasing approaches) has a fairly good prediction result (acc: 80%), and does not have the F1 score problem as seen in FairGNN. Additionally by applying some of the debiasing approaches we can see some slight improvement in some fairness metrics (e.g. EOD metric), while still having a comparable prediction accuracy (e.g. sample approach).

	Performance		Fairness			
Models	Acc	F1	Δ_{SPD}	Δ_{EOD}	Δ_{OAED}	Δ_{TED}
FairGNN	0.64 ± 0.003	0.395 ± 0.003	**0.002 ± 0.004**	**0.002 ± 0.014**	**0.005 ± 0.006**	**0.002 ± 0.010**
RHGN	0.728 ± 0.007	0.678 ± 0.011	0.011 ± 0.005	0.026 ± 0.019	0.021 ± 0.020	0.022 ± 0.011
RHGN (S)	**0.734 ± 0.004**	0.70 ± 0.003	0.012 ± 0.005	0.037 ± 0.011	0.034 ± 0.013	0.027 ± 0.017
RHGN (R)	0.728 ± 0.007	0.678 ± 0.011	0.011 ± 0.005	0.026 ± 0.019	0.017 ± 0.008	0.022 ± 0.011
RHGN (D)	0.646 ± 0.047	0.446 ± 0.123	0.003 ± 0.006	0.007 ± 0.120	0.053 ± 0.004	0.011 ± 0.021
CatGCN	0.720 ± 0.003	**0.704 ± 0.004**	0.045 ± 0.008	0.051 ± 0.007	0.046 ± 0.012	0.175 ± 0.037
CatGCN (S)	0.711 ± 0.052	0.667 ± 0.112	0.035 ± 0.014	0.052 ± 0.022	0.067 ± 0.030	0.128 ± 0.100
CatGCN (R)	0.598 ± 0.093	0.576 ± 0.088	0.024 ± 0.007	0.043 ± 0.013	0.063 ± 0.034	0.058 ± 0.025
CatGCN (D)	0.621 ± 0.050	0.591 ± 0.018	0.033 ± 0.004	0.065 ± 0.010	0.042 ± 0.022	0.050 ± 0.011

Table 5.9: Comparison of the results using different models from the framework on the **JD** dataset.

- Finally by observing the CatGCN model results, we can see a similar prediction accuracy on the original model (without debiasing approaches) in comparison to the RHGN model. Unfortunately by applying the different debiasing techniques, we can see no huge improvement in both the model prediction accuracy and fairness metrics using all the debiasing approaches.

In the end, we can see again as in the previous sections that the debiasing approaches results are different when applied on different models.

5.6 JD

The fourth and last experiment is testing the framework on the JD dataset. This dataset was also used alongside with the Alibaba dataset in the original RHGN and CatGCN paper experiments. Additionally the authors in [] have also shown that training both RHGN and CatGCN on this dataset did not result in fair predictions. As in the previous experiments, we first train the original models without applying any debiasing approaches for 5 runs, and then do the same for the different debiasing techniques. JD is the second dataset that will be new for the FairGNN model, since it was trained on in the original paper. Table 5.9 shows the experiment results.

From the results in Table 5.9. we can make the following observations:

- Training FairGNN on the JD dataset does not give us better results in comparison to RHGN and CatGCN. In addition we can also observe that we have the same problem regarding the model prediction accuracy and the F1 score. Similarly as in Alibaba dataset, a reason for this can be that the model is concentrating only on predicting a specific class and ignoring the other. This has as well a huge effect on the fairness metrics results similar as in the Alibaba dataset. Again this is a preliminary experiment, and we believe that FairGNN can perform better on this dataset if this problem can be mitigated.

- In comparison to FairGNN results, the RHGN model (without debiasing approaches) performs much better in regards to prediction accuracy and F1 score (acc: 72% and F1: 67%). Also when applying the different debiasing approach we can observe a slight improvement in the model prediction accuracy. For example using the sample approach we obtain a prediction accuracy of about 73%, but with no huge improvement in the fairness metrics. Similar to the Alibaba dataset, the disparate impact remover approach does not give us better prediction accuracy, nor better fairness metrics results.

- Lastly, the original CatGCN trained without any debiasing approaches gives us comparable results as in RHGN (acc: 72% and F1: 70%). Using the different debiasing approaches on the model, we have seen only a slight decrease in the SPD metric (SPD: 0.024), but with a loss in the model prediction accuracy (acc: 59%) in comparison to the original CatGCN model. We can also observe that using the disparate impact remover approach did not decrease the model prediction accuracy dramatically in comparison to the RHGN model results, which can suggest that the disparate impact debiasing approach is not effective for this dataset.

To conclude from the presented results, we can observe in general how each model behaves differently on the JD dataset in terms of prediction accuracy and fairness metrics. We can also see how FairGNN behaves on the newly presented dataset (JD) and from the observed preliminary results, we believe that there is a room for improvement. Additionally in general from the presented results using the different datasets we can observe for example how the reweighting debias approach can help in increasing the prediction accuracy of the model, while maintaining a comparable fairness metrics results as the original model (e.g. CatGCN). Also, we can see from the results in general, that the disparate impact remover was the most ineffective approach when it comes to prediction accuracy improvement. At the end, there is always be a trade-off between prediction accuracy and fairness metrics, but from the experiments we have showed at least that there are pre-processing debias approach that can help in improving both metrics (accuracy and fairness metrics). This can give the user a good insight in the future using the framework on deciding which model or which debias approach can be best combined if needed to obtain the best results for his/her use case.

So I realised that I was just a person, and I get another shot at it, every day the sun comes up.

Bill Murray

6 Conclusion and Future Work

In this thesis, we proposed and implemented a novel framework for fairness analysis and mitigation for Graph Neural Networks based on user profiling classification. The new system is a standardised version for training several GNN models sequentially in order to have a better understanding and better analysis of the prediction and the model fairness results. With the help of the introduced framework we were able to show that we can have such a system that can be used publicly for users in the research community to analyse GNN and fairness results. We first began this thesis with a literature review about several topics that are related to our novel framework like user profiling, graph neural networks and fairness. The goal for such review was to introduce theses topics to the reader, and present the current state of the art approaches for each of these topics, especially for GNN models and debiasing approaches. We then introduced the needed preliminary knowledge that we used for implementing the novel framework. We first introduced the fairness metrics that we used, namely in terms of disparate impact and disparate mistreatment. For evaluating disparate impact value of the analysed models we used *statistical parity*, *equal opportunity*, and *overall accuracy equality*. We also introduced the *treatment equality* metric to evaluate the disparate mistreatment. Additionally introducing the different GNN models that we have used in our framework and our experiments, namely *FairGNN*, *RHGN*, and *CatGCN*. Finally, we also introduced the debiasing approaches that were suitable to be used for our framework which were all pre-processing debiasing approaches such as *reweighting*, *sampling*, and *disparate impact remover*.

After introducing the preliminary knowledge, we presented the framework structure which is divided into 3 components, which are the *Pre-processing component*, the *Debaising component*, and the *Core component*. The most important component in our framework was the pre-processing component which tries to make sure that the input data from the user is processed correctly for each model in the System respectively so that we can train the input data on all different models. We then introduced the debiasing component which applies the user chosen debias approach if needed on the dataset before training. Finally we presented the core component structure and main goal, which was to train the different models using the data that was pre-processed sequentially.

After designing and implementing the novel framework we conducted several experiments to evaluate the effectiveness of it. We first tested if the framework is capable of training several models sequentially using the same dataset. For this test, we have built a simple web app based to make it easier for the user to use the whole system and analyze the results. From the experiment, we have observed that our novel framework can indeed preprocess the input data correctly and train the models using the processed data. Then we conducted a user study using the models we have implemented, to test the effectiveness of the different debiasing approaches in comparison to the state of the art GNN fairness model which is FairGNN. The user study was done using the three debiasing approaches and the four datasets that were all introduced in chapter 3. We then tried to train each model with each dataset and also using every debiasing approach combination. Our first experiment in the user study was using the NBA dataset, according to the presented results we have observed that FairGNN had the most stable results in term of accuracy and fairness metrics compared to the other models and debiasing approaches. Nonetheless, since both RHGN and CatGCN model have several hyperparameters to fine tune, we believe that by applying a different hyperparameter selection approach the user can achieve a much better results for both models. The second experiment was conducted on the pokec-z dataset. From the conducted experiments we mainly observed that RHGN does not perform good on this dataset. A reason for this can be due to model overfitting or a problem in the optimization approach that was used in RHGN model per default which makes the model get stuck in a local minima. In all experiments we did not change any of the optimization approach that was used originally by the models. This can be a solution for future work to adjust or change the optimization function that the RHGN model is using to have better results in regards to the pokec-z dataset. Another reason can be also due to the hyperparamter tuning that was used, which also can be improved in future work. Other than this, the CatGCN model showed us comparable results to FairGNN, with also being better in term of fairness when applying some of the debiasing approaches. Then we conducted the third experiment on the alibaba dataset. We have clearly observed how bad the results were for FairGNN when trained on this dataset. The main problem was the huge difference between the model prediction accuracy and F1 metric, which also makes the fairness metrics results invalid for this experiment. We believe that since the alibaba dataset is composed of different datasets (e.g. user, ads, and clicks) by joining them together into one dataset, we lose many features that can be helpful in the model training. Since both RHGN and CatGCN pre-processing approach take advantage of the Alibaba dataset structure they get better results in comparison to FairGNN pre-processing approach. A solution for this problem can be adjusting the FairGNN pre-processing approach that was originally published to accommodate such datasets so that no features are lost. Other than that, both RHGN and CatGCN models showed comparable results in the prediction accuracy, with RHGN having a slight improvement in the fairness metrics when using some of the debiasing approaches. Finally we trained the models using the JD dataset. We got as well the same problem regarding model accuracy prediction and F1 metric being vastly different in regards to the FairGNN model. The same hypothesis presented while training on alibaba dataset applies also here. Nonetheless both the RHGN and CatGCN models had similar results in terms of accuracy and fairness metrics when applying the different debiasing approaches.

We see two main directions for future work. First, are the possible improvements of the already built System that we have presented. The first improvement that we see, is having an option to support multi-class classification for the framework. This of course will lead to adding new models that supports multi-class classification in GNNs. Additionally, adjusting the pre-processing component to accommodate the newly added models. Similarly, adding other debiasing approaches in the framework. These approaches can also support both binary classification as well as multi-class classification. Since we only conducted our experiments using pre-processing debiasing approaches, it would be also very beneficial to examine how the models will behave if we can add other debiasing approaches (e.g. in-processing approaches or post-processing approaches) without altering the model structure. Another idea for a future work can also be trying to implement the same debiasing approach used in FairGNN to other models and examining the results using this approach. In our work we could not achieve this with our models, since it will alter the main structure of these models as discussed, but this can be achievable when adding models that can accommodate the in-processing debiasing approach used in FairGNN.

Furthermore, improving the results of the models that were presented in this work. This can for example be done using different hyperparamater tuning approach other than what we used to fix the problems we have encountered with some of the models results. Also using different optimization methods to solve the problem of the local minima that we have seen for example in the pokec-z dataset with the RHGN model. Finally, the user-interface that was built can be also improved, to accommodate different options as well as introducing new features when extending the novel framework. One can also think of adding some better results visualization in the user-interface to improve the user experience and help the user in analysing the results efficiently.

Bibliography

[1] Golnoosh Farnadi et al. “User profiling through deep multimodal fusion”. In: *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*. 2018, pp. 171–179.

[2] Delip Rao et al. “Classifying latent user attributes in twitter”. In: *Proceedings of the 2nd international workshop on Search and mining user-generated contents*. 2010, pp. 37–44.

[3] Thomas N Kipf and Max Welling. “Semi-supervised classification with graph convolutional networks”. In: *arXiv preprint arXiv:1609.02907* (2016).

[4] Shu Wu et al. “Session-based recommendation with graph neural networks”. In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 33. 01. 2019, pp. 346–353.

[5] Weijian Chen et al. “Semi-supervised User Profiling with Heterogeneous Graph Attention Networks.” In: *IJCAI*. Vol. 19. 2019, pp. 2116–2122.

[6] Qilong Yan et al. “Relation-aware Heterogeneous Graph for User Profiling”. In: *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*. 2021, pp. 3573–3577.

[7] Weijian Chen et al. “CatGCN: Graph Convolutional Networks with Categorical Node Features”. In: *IEEE Transactions on Knowledge and Data Engineering* (2021).

[8] Xiangnan He et al. “Lightgcn: Simplifying and powering graph convolution network for recommendation”. In: *Proceedings of the 43rd International ACM SIGIR conference on research and development in Information Retrieval*. 2020, pp. 639–648.

[9] Rex Ying et al. “Graph convolutional neural networks for web-scale recommender systems”. In: *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*. 2018, pp. 974–983.

[10] Liang Yao, Chengsheng Mao, and Yuan Luo. “Graph convolutional networks for text classification”. In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 33. 01. 2019, pp. 7370–7377.

[11] Ninareh Mehrabi et al. “A survey on bias and fairness in machine learning”. In: *ACM Computing Surveys (CSUR)* 54.6 (2021), pp. 1–35.

[12] Erasmo Purificato, Ludovico Boratto, and Ernesto William De Luca. “Do Graph Neural Networks Build Fair User Models? Assessing Disparate Impact and Mistreatment in Behavioural User Profiling”. In: *Proceedings of the 31st ACM International Conference on Information & Knowledge Management*. 2022, pp. 4399–4403.

[13] Enyan Dai and Suhang Wang. “Say no to the discrimination: Learning fair graph neural networks with limited sensitive attribute information”. In: *Proceedings of the 14th ACM International Conference on Web Search and Data Mining*. 2021, pp. 680–688.

[14] Sue Henczel. “Creating user profiles to improve information quality”. In: *Online (Weston, CT)* 28.3 (2004), pp. 30–33.

[15] Sara Rosenthal and Kathleen McKeown. “Age prediction in blogs: A study of style, content, and online behavior in pre-and post-social media generations”. In: *Proceedings of the 49th annual meeting of the association for computational linguistics: human language technologies*. 2011, pp. 763–772.

[16] Takuo Hamaguchi et al. “Knowledge transfer for out-of-knowledge-base entities: A graph neural network approach”. In: *arXiv preprint arXiv:1706.05674* (2017).

[17] Yiwei Sun et al. “Node injection attacks on graphs via reinforcement learning”. In: *arXiv preprint arXiv:1909.06543* (2019).

[18] Xianfeng Tang et al. “Transferring robustness for graph neural network against poisoning attacks”. In: *Proceedings of the 13th international conference on web search and data mining*. 2020, pp. 600–608.

[19] Wei-Lin Chiang et al. “Cluster-gcn: An efficient algorithm for training deep and large graph convolutional networks”. In: *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*. 2019, pp. 257–266.

[20] Petar Veličković et al. “Graph attention networks”. In: *arXiv preprint arXiv:1710.10903* (2017).

[21] Joan Bruna et al. “Spectral networks and locally connected networks on graphs”. In: *arXiv preprint arXiv:1312.6203* (2013).

[22] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. “Convolutional neural networks on graphs with fast localized spectral filtering”. In: *Advances in neural information processing systems* 29 (2016).

[23] Qimai Li, Zhichao Han, and Xiao-Ming Wu. “Deeper insights into graph convolutional networks for semi-supervised learning”. In: *Thirty-Second AAAI conference on artificial intelligence*. 2018.

[24] Thomas N Kipf and Max Welling. “Semi-supervised classification with graph convolutional networks”. In: *arXiv preprint arXiv:1609.02907* (2016).

[25] Will Hamilton, Zhitao Ying, and Jure Leskovec. “Inductive representation learning on large graphs”. In: *Advances in neural information processing systems* 30 (2017).

[26] Tianchi Yang et al. "HGAT: Heterogeneous graph attention networks for semi-supervised short text classification". In: *ACM Transactions on Information Systems (TOIS)* 39.3 (2021), pp. 1–29.

[27] Faisal Kamiran and Toon Calders. "Classifying without discriminating". In: *2009 2nd international conference on computer, control and communication*. IEEE. 2009, pp. 1–6.

[28] Michael Feldman et al. "Certifying and removing disparate impact". In: *proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining*. 2015, pp. 259–268.

[29] Ian Goodfellow et al. "Generative adversarial networks". In: *Communications of the ACM* 63.11 (2020), pp. 139–144.

[30] Prasanna Sattigeri et al. "Fairness GAN: Generating datasets with fairness properties using a generative adversarial network". In: *IBM Journal of Research and Development* 63.4/5 (2019), pp. 3–1.

[31] Geoff Pleiss et al. "On fairness and calibration". In: *Advances in neural information processing systems* 30 (2017).

[32] Cynthia Dwork et al. "Fairness through awareness". In: *Proceedings of the 3rd innovations in theoretical computer science conference*. 2012, pp. 214–226.

[33] Muhammad Bilal Zafar et al. "Fairness beyond disparate treatment & disparate impact: Learning classification without disparate mistreatment". In: *Proceedings of the 26th international conference on world wide web*. 2017, pp. 1171–1180.

[34] Michael Feldman et al. "Certifying and removing disparate impact". In: *proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining*. 2015, pp. 259–268.

[35] Michael Feldman et al. "Certifying and removing disparate impact". In: *proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining*. 2015, pp. 259–268.

[36] *Streamlit*. URL: `https://streamlit.io/`.

[37] *Neptune*. URL: `https://neptune.ai/`.

Declaration of Authorship

I hereby declare that I have written the present work myself and did not use any sources or tools other than the ones indicated.

Mohamed Abdelrazek

Magdeburg,
March 8, 2021