RECOMMENDING WHOM TO FOLLOW ON GITHUB

A Thesis
Submitted to the Graduate Faculty
of the
North Dakota State University
of Agriculture and Applied Science

By

Muhammad Usman Sarwar

In Partial Fulfillment of the Requirements
for the Degree of
MASTER OF SCIENCE

Major Department:
Computer Science

November 2021

Fargo, North Dakota

NORTH DAKOTA STATE UNIVERSITY

Graduate School

Title

RECOMMENDING WHOM TO FOLLOW ON GITHUB

By

Muhammad Usman Sarwar

The supervisory committee certifies that this thesis complies with North Dakota State University's regulations and meets the accepted standards for the degree of

MASTER OF SCIENCE

SUPERVISORY COMMITTEE:

Dr. Muhammad Zubair Malik
Chair

Dr. Saeed Salem

Dr. María de los Ángeles Alfonseca-Cubero

Approved:

11/19/2021
Date

Dr. Simone Ludwig
Department Chair

ABSTRACT

Social networking activities on GitHub allow the construction of interesting interaction networks. One such network is 'follow-network' which enables an effective information dissemination process. As a result, GitHub users are bombarded with stacks of information which also puts the users at risk of information overload. This motivates us to recommend the relevant user such that developers are only provided with the relevant information. In this work, we present an attributed network embedding based framework to recommend whom to follow on GitHub. This is a challenging task due to the complex social network structure of the developers. In particular, we first construct a developers' 'follow-network'. Further, we extract the node embeddings of each node and feed these embeddings to a K-nearest Neighbour classifier. We validate our approach on the developers of three popular programming languages (C++, Python, and Java). We were able to achieve promising results with an F1-score of 72%.

ACKNOWLEDGEMENTS

Throughout my 2 years at NDSU as a Master's student, I have been blessed to have wonderful people around me. I have received great support from all my peers, colleagues, and professors. First, I would like to thank Dr. Muhammad Zubair Malik. He has been consistently supporting me throughout my tenure at NDSU. Even during the early Covid-19 days, he has consistently supported me. I believe his guidance and supervision pushed me to sharpen my critical research thinking and brought my research work to a higher level. I would also like to thank Dr. Saeed Salem who is also my co-advisor. I have been very fortunate to meet Dr. Saeed Salem. He has been guiding me throughout my tenure at NDSU not only in academic matters but also in general life matters. I am thankful to several funding agencies that supported my research, including ND EPSCoR. Finally, I want to thank Sarim Zafar for his constant support throughout. He has been a great friend and colleague. I always look up to him for most of my life decisions.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1. INTRODUCTION

1.1. Overview

GitHub [1] is the world's largest version control and source code management service used by more than 50 million developers worldwide. Three million businesses and organizations including Microsoft, Facebook, Google, IBM, Uber, and Walmart use it for various projects. It has more than 100 million publicly available repositories [2], which can be mined for analysis through their APIs. This has attracted the researchers from industry and academia to explore various types of analysis around GitHub dataset such as exploring the social patterns [1], analyzing the toxicity in Github developers [3] community [2, 3], and extracting authorship of the source code [4, 5].

Various kinds of social coding platforms such as GitHub [4], GitLab [5], BitBucket [6] allow us to develop social ties that help the developers collaborate and work together. Such social relations can help in improving the collaborations in the software development [1]. GitHub employs several social media activities such as follow, watch, and fork which helps in collaborations among the software developers. Using such social media activities, many interesting interaction networks of the developers can be constructed. One such network could be 'follow-network', where each node [7] represents a user and the edge between the two nodes represents the following activity between them. By analyzing such interaction networks, we can answer many interesting questions such as who are they connected to in a massive community? who is the influential developer in the community? what is their role in specific software development projects?

Conventional social media platforms such as LinkedIn [8], and Facebook [9] provides the opportunity to connect to the users who share similar characteristics such as hobbies, work interests, and similar friend circle. However, the connections in such conventional social media platforms

[1] https://github.com/
[2] https://venturebeat.com/2018/11/08/github-passes-100-million-repositories/
[3] Throughout the document we use 'developer' and 'user' interchangeably.
[4] https://github.com/
[5] https://about.gitlab.com/
[6] https://bitbucket.org/product
[7] nodesandverticesareusedinterchangeablyinthisdocument
[8] https://www.linkedin.com/
[9] https://www.facebook.com/

are represented as reciprocal links that result in an undirected graph [10]. However, GitHub follow relationships are directional in nature, hence the resultant graph is a directed graph. A directed relationship allows the users to follow other users without requiring them to reciprocate. Additionally, undirected social networks naturally allow some users to be followed by many people without following many themselves, effectively becoming “celebrities” or “stars” [6]. Such undirected relationships are based on information dissemination interest by the follower rather than purely social interactions. Typically, social media platforms recommend the list of users to follow similar to Facebook’s "People You May Know" feature. This recommendation is usually generated based on mutual connections. This works well in an undirected graph, primarily due to the undirectional nature of the relationship. However, in disciplines such as software engineering, follow-network provides effective dissemination of knowledge through informal relations and information interest. GitHub users are bombarded with stacks of information. Also, such information bombardment puts them at risk of information overload, which motivates us to recommend the relevant user such that developers are only provided the relevant information.

Numerous studies [6, 7, 8, 9] have been conducted to recommend the followers in social networks specific to software engineering. However, there is less focus on recommending the followers in directed network [6, 7]. Additionally, existing approaches suffer from two types of limitations: a) they only consider structural features of the interaction network to recommend whom to follow b) they are mostly focused on recommending influential users based on homophily to recommend whom to follow. However, social network platforms have access to much more information than the network structure such as interests, hobbies, and educational background. Also, users provide clues about what interests them by commenting and reacting to social posts. To address the aforementioned issues, we used the attributed-graph based embedding approach to learn a deep representation that maps each developer to a fixed-length vector. Recently, embedding extraction methods have been quite popular due to the ability to incorporate the structural as well as attribute properties of the network. Further, these extracted embedding vectors can be used to solve downstream tasks such as node classification, link prediction, and clustering. The attributed-graph based embedding method learns the meaningful embedding vectors such that similar developers will have embedding vectors close to each other and dissimilar developers will have embedding vectors

[10]Term graph and network are used interchangeably

farther apart from each other when mapped on the embedding space. In this paper, we present an attributed network embedding based framework to recommend whom to follow. We first constructed an interaction graph i.e. a follow-graph to encode the 'follow' relationships between the GitHub users. Further, the follow-graph is fed to the attributed embedding extraction method that maps each node into a fixed-length vector representation. The attributed embedding method confines the similar developers 'close' to each other and keeps the dissimilar developers separated from each other in the embedding space. Further, these representations are fed to the K-nearest Neighbour (KNN) classifier to recommend whom to follow. The primary advantage of utilizing a nearest neighbor classifier is its lazy learning capability and its ability to incorporate new samples without the need to retrain. Our study attempts to answer the following research questions.

1. RQ 1: Can we predict whom to follow on GitHub by incorporating the user centered features such as similarity based features along with the network structural features?

2. RQ 2: How does our approach perform on the developer with low interaction?

To address the aforementioned research questions, we evaluated our framework on the GitHub developers who code in the three most popular programming languages: C++, Java, and Python on GitHub. We were able to achieve promising results with the F1-score of 72% by utilizing attributed embedding vector, which is an order of magnitude better than the results obtained by using network structure based embedding vectors i.e. F1-score of 67%. To the best of our knowledge, this is the first study to recommend whom to follow on GitHub using the attributed node embedding based approach. Our framework has immediate benefits on the industry, academia, and open-source community: it can help the new or less experienced developers to follow the right developer based on numerous features such as shared research interests, hobbies, friend circles, and behavior. Also, the strong follower network will enable the users to come up with better collaborations and such collaborations will help in improved software development processes and better software products.

1.2. Contribution and outline

In Chapter 2, we discuss the previous research studies that are relevant to this work. In Chapter 3, we discuss the theoretical background for node embedding vector and other related concepts such as Link Prediction and Supervised Learning. This section will help in establishing the concepts required to understand our research work. In Chapter 4, we formally define our

research problem. Further, we discuss the proposed framework along with the different stages i.e. Data Collection, Interaction Graph. Embedding Extraction, and Link Prediction (Supervised Classification). In Chapter 5, we evaluate our proposed framework using 815420 GitHub users. In Chapter 6, we discuss the potential future avenues of our research work. Finally, in Chapter 7, we conclude our research work.

2. RELATED WORK

This chapter discusses the existing work across three different research areas: (1) GitHub Social Analysis (2) Whom to Follow (3) Attributed Network Embedding Extraction. Existing studies on the GitHub social analysis using the GitHub dataset are discussed in section 2.1, whom to follow existing studies are discussed in section 2.2, and finally attributed network embedding existing studies are discussed in section 2.3.

2.1. GitHub social analysis

Numerous studies have been conducted to explore the social space of GitHub. These studies range from exploring social behavior to quantifying the influence and expertise of the GitHub developer. Yu et al. [1] explored the social behavior among different software developers on GitHub. This study can provide instructions to the developers who are looking for collaborations. They addressed multiple research questions. First, they explored the growth of GitHub with three different OSS communities i.e. Freecode [1], Alioth [2], Savannah [3]. Then, they constructed the follow-network based on the follow behavior among the developers. Finally, they analyzed the social behavior patterns such as independence, star, and group patterns respectively among the developers by analyzing the follow-networks. They also analyzed the effect of the leader node, herd behavior.

Heller et al. [10] applied the visualization technique to the GitHub user profiles and repositories data. They explored this data to identify the patterns (effect of geographic distance, social connectivity, etc) within the software development community. They utilized numerous visualization approaches: geo-scatter maps, small multiple displays, and matrix diagrams. These methods are not intended to provide conclusive answers; instead, they provide a way for researchers to explore the question space and communicate initial insights. They also released a tool [4] for public use, where researchers can use the collected data and find their own patterns – using the same interactive web-based visualization.

[1] `https://www.freecodecamp.org/`
[2] `http://alioth.debian.org/`
[3] `http://savannah.gnu.org/`
[4] `http://gothub.stanford.edu`

Hu et al. [11] analyzed the user influence using the social interactions on GitHub. They have considered the social relationships between users, relations between users and projects, and the activities. Specifically, they proposed a following, star, and Fork activity-based approach to measure user influence in the GitHub social network. Further, they adopted HITS [12] , PageRank [13] and H-index [14] to measure the influence of the given user from the different perspective and used the Borda count [15] to obtain a comprehensive measure of the influence of each user. Here, they analyzed the user influence in the social network in terms of popularity, centrality, content value, contribution, and activity.

Fu et al. [16] analyzed the nature of interactions on GitHub. First, they built an interaction graph using the GitHub Events i.e. 'Watch' Events, 'IssueComment' Event, and 'PullRequest' to represent the main user interactive behaviors. They end up with a graph of 6,012,074 nodes and 42,700,112 edges. Secondly, they analyzed the important users such that users play a critical role in information dissemination. To illustrate the relationship between inter-community interactions and to explore the role of important users, they employed structural hole theory [17]. Specifically, they applied two structural hole (SH) spanner algorithms i.e. Constraint [17] and HIS [18] on the interaction graph to find the SH spanners. They found SH spanners to have significant importance in the graph such that they have larger degrees, higher centrality, and a more connected community structure. Also, SH spanners have more social connections on GitHub as compared to the average GitHub user and they tend to have more complete GitHub profiles.

Dey et al. [19] presented a framework to extract the representation of developer expertise. They constructed a Skill space, where each API, developer, and project is represented as a vector representation. Specifically, they used Doc2Vec [20] embedding extraction method to embed the API, developer, and project as a vector representation. Here, they have used World of Code (WoC) [21] to extract the API used by the developers of 17 programming languages. Also, they evaluated how the developer's representation vector aligns with their reported API expertise. They were able to extract interesting insights. For instance, developers usually use APIs that are closely related to their expertise. Also, developers tend to join a project if it is more closely aligned to their expertise.

Montandon et al. [22] utilized an unsupervised and supervised machine learning (ML) based classifier to identify the expert in the three popular Javascript libraries i.e. 'facebook/react', 'mongodb/node-mongodb' and 'socketio/socket.io'. First, they collected 12 features regarding the

GitHub user activity on the GitHub project. These features include but are not limited to a number of commits on the source code file that import that library, number of projects GitHub user contributed to. To establish the ground truth, they also surveyed a sample of GitHub users who work with the candidate libraries. In total, they surveyed 575 GitHub developers including 418 'facebook/react' developers, 68 'mongodb/node-mongodb' developers, and 89 'socketio/socket.io' developers. They concluded that supervised standard Ml models (Random Forest and SVM) did not perform well on this problem. In contrast, they proposed an unsupervised method to identify the library expect based on their similarity to a defined cluster. Here, the similarity is calculated based on the features associated with each GitHub developer.

Dabbish et al. [23] explored the value of transparency for large-scale collaborations and communities within the GitHub user-base. To be precise, they examined how GitHub developers utilize the information about other developers' actions on the repository. To achieve the goal they interviewed a number of GitHub users ranging from novice to expert developers. They found interesting insights regarding the GitHub users interactions, with such social interactions we can infer other developers' goals, and also we can predict if a project has the chance to thrive in long term or not. Also, developers utilize such insights for coordinating in projects, strengthening their technical skills, and managing their reputation. Such work can help in community building, information dissemination using different social actions such as by collaborating on different projects.

2.2. Whom to follow

Numerous studies [6, 7, 24, 25] have been conducted to recommend the followers in social networks. However, there is little focus on recommending the followers in directed network [6, 7]. Additionally, existing approaches suffer from two types of problems: a) they only consider structural features of the network to recommend whom to follow b) they are mostly focused on recommending influential users based on homophily to recommend whom to follow. Social network services have access to much more information than the network structure: users supply details about their interests and background, and they provide clues about who interests them by choosing whose posts to read and respond to. This led us to ask what features in a user's profile, behavior, and network might be most effective in recommending people. To address the aforementioned issues, we used an attributed-graph based embedding approach to learn a deep representation that maps the node to a fixed-length vector. The attributed-graph based embedding method would learn the

vectors such that similar users will have embedding vectors close to each other and dissimilar users will have vectors farther apart from each other in the embedding space.

Schall et al. [6] presented a 'who to follow' approach that is based on the concept of user authority using techniques such as HITS [12], and PageRank [13]. The approach is based on the network analysis techniques primarily based on the authority metric w.r.t to a specific area. They measure the user authority based on the activity such as repository commit message, community reputation such as follower degree count. The presented concepts have been evaluated using a real-world dataset. They presented a detailed comparison between personalized and non-personalized recommendations and shows that personalized recommendations are comparatively better than non-personalized recommendations.

Zhu et al. [26] proposed a link prediction method based on network structure and topic distribution (NSTD). The methods utilized the information from the user-generated content and structural features of the social networks. First, they incorporated the inherent features of the social network i.e. homophily, transitivity, clustering, and degree heterogeneity for the directed follower network. Further, they Incorporated a similar measurement using the topic distribution model i.e. LDA on user generated content. They evaluated their approach on Sina Weibo [5], a popular social media platform in China, and reported an 87% F1-score.

Sharma et al. [9] proposed a two-stage classification approach to generate recommendations on Whom to follow on Twitter. First, they filtered and labeled the Twitter users that are potentially interested in software development. Secondly, they extracted different types of features i.e., content, network, profile, and GitHub features to characterize a Twitter user, and then these features are further fed to the two-stage classification approach to generate a discriminative model, which has the capability to differentiate specialized domain software gurus and other Twitter user with respect to each domain. They evaluated their approach on a dataset of 86,824 Twitter users across four different domains (JavaScript, Android, Python, and Linux) and reported an F1-score of 82%.

Blincoe et al. [25] studies the motivation behind following the developers and they also studied the influence of popular users on their followers. Mixed methods research approach was used including a survey of 800 GitHub users to uncover the reasons for following on GitHub and complementary quantitative analysis of the activity of GitHub users to examine influence. They

[5] www.weibo.com

evaluated 199 most followed users along with their followers. They found the popular users to have a significant influence on their followers. Popular users often attract their followers to a new project. This open collaboration of GitHub is shaping a new social structure in open source projects.

Mohan et al. [24] proposed the method to find out which members are more influential than the rest. They extracted each user's data and identify 'ego' using the PageRank algorithm. Further, these 'ego' are used to recommend new followers using clusters. Here, the ego represents a developer/users linked to everyone in the network, and 'alters' represent other users in the network.

Gelley et al. [8] examined the utilization and effectiveness of the Pinterest follow mechanism for content dissemination. They mostly discussed the user interaction characteristics associated with the activity network by answering the research question such as the effectiveness of Pinterest follow for content discovery, Is the follow-interaction is based on interest homophily. To answer, such a question, they compared the implicit activity graph with the follow graph. They conclude that the follow behavior is not significantly utilized in content sharing on Pinterest.

Jiang et al. [27] provided an analysis on the factors associated with the unfollowing behavior in GitHub using the follow relationship of 701,364 developers. In addition, they also conducted a survey to understand the reasons for unfollowing behavior and then used a logistic regression model to analyze the correlation of various factors associated with unfollowing behavior. They found that developers unfollow the developers who are not that active i.e. with fewer activities. Also, the programming language similarity is one of the major contributors in follow activity, so developers unfollow other developers who do not work on the same programmer languages. Finally, the reciprocal follow relationship makes the bond strong and there is less chance that they unfollow.

2.3. Attributed network embedding extraction

Network embedding extraction and link prediction in directed networks have been a hot topic in many domains including network science, information system, and data mining. Primarily, network embedding has become an efficient approach to deal with large-scale networks. Numerous network embedding extraction methods are proposed ranging from non-attributed [28, 29, 30] to attributed-based embedding. Here, non-attributed embedding only considers network topological features while attributed embedding considers node-related attributes along with the network topological features. Specifically, attributed embedding has become increasingly popular due to its state-of-the-art results on the baseline datasets. We have discussed some of the popular attributed

embedding extraction methods. We also discussed a few studies which address the link prediction problem in the graph/network.

Deng et al. [31] also proposed an attributed node embedding framework. The multi-level framework called GraphZoom improves both the accuracy and scalability of unsupervised graph embedding algorithms. it consists of four major kernels (1) graph fusion (2) spectral graph coarsening (3) graph embedding (4) embedding refinement. GraphZoom first performs graph fusion to generate a new graph that effectively encodes the graph topology and the node attributes information. Then, the generated fused graph is repeatedly broken down into smaller graphs by merging nodes with high spectral similarities. The coarsening algorithm retains the first few eigenvectors of the graph Laplacian matrix to make sure the key graph structure is preserved. Then, during the graph embedding kernel any existing unsupervised graph embedding techniques can be applied to obtain node embeddings for the graph at the coarsest level. Lastly, the embedding refinement algorithm has employed the embeddings back to the original graph by applying a proper graph filter to ensure embeddings are smoothed over the graph. They evaluated the approach on a number of popular baseline graph datasets i.e. Cora, Citeseer, and Pubmed citation network and PPI, and Reddit.

Hamilton et al. [32] proposed GraphSage, an inductive framework that leverages node features to generate the node embedding on the previously unseen nodes. it is based on the unsupervised loss function that allows being trained without task-specific supervision. Also, instead of training distinct embedding vectors for each node, they trained a set of aggregators functions that learn to aggregate information from the candidate's local neighborhood. Here, each aggregator function aggregates information from a different number of hops from the given. GraphSage was evaluated on three inductive datasets (Citation, Reddit, and PPI), and based on evaluation results it outperforms the existing state-of-the-art methods.

Zhang et al. [33] addressed the issue of missing data and scalability in the existing attributed node embedding methods. In particular, they proposed an attributed node embedding method called Scalable Incomplete Network Embedding (SINE) for learning node representation from the incomplete graph. They trained a 3-layered neural network with nodes represented as a one-hot embedded vector. Also, they derive an online optimization strategy based on the stochastic gradient descent which enables the SINE to be efficient and scalable for large real-world networks. They evaluated the effectiveness of SINE on number of real-world graphs i.e. Cora, Citeseer, DBLP(Subgraph)

[6] and DBLP(Full) [7]. They showed that SINE outperforms the multiple state-of-the-art methods (DeepWalk [28] and node2vec [30]) on graphs with missing links, and node attributes.

Previously discussed embedding extraction method can only extract node representation embedding given the network is undirected in nature. In the subsequent paragraphs, we have discussed the attributed network extraction methods that have the ability to extract embedding vectors in undirected and directed networks. Huang et al. [34] proposed a distributed scalable attributed embedding method called 'Accelerated Attributed Network Embedding' (AANE). It is a scalable and efficient embedding extraction method that can learn embedding vectors by incorporating the network structural properties as well as nodes' attributes proximity. AANE enables the joint learning process in a distributed manner by decomposing the complex problem into multiple subproblems. Specifically, they learned an N-dimensional representation based on the decomposition of the attributed similarity and structural embedding differences between the connected nodes. They evaluated the approach using a number of real-world datasets i.e. 'BlogCatalog', 'Flickr', and 'Yelp'. They compared AANE with three classical node embedding baseline method (DeepWalk [28], PCA [35], and Line [29]) and two attributed node embedding method (LCMF [36] and MultiSpec [37])

Liao et al. [38] also proposed a neural network based attributed node embedding based framework for learning effective node representation. To incorporate the complex interaction between nodes, where nodes also have features associated with it, they have used a multi-layer neural network. the last layer has the softmax activation function which outputs the probability vectors. Also, the neural network model is trained using the Adaptive Moment Estimation (Adam) optimizer. They evaluated the method on the four public datasets i.e. two Facebook networks, DBLP [8] and CITESEER [9]. They found their method to be effective as compared to the existing state-of-the-art baselines such as node2vec [30], LINE [29]. TriDNR [39], and AANE [34]. We have used ASNE and AANE as our embedding extraction methods. Section 3.1.1 discussed this attributed network embedding approach in detail.

Khosla et al. [40] also proposed an approach for learning the node representations specifically in a directed graph called Node Embeddings Respecting Directionality (NERD). NERD can be

[6] `https://www.aminer.org/citation`
[7] `https://www.aminer.org/citation`
[8] `http://arnetminer.org/citation`
[9] `http://citeseerx.ist.psu.edu/`

applied to any type of graph i.e. directed, undirected, weighted, and unweighted. NERD uses the alternating random walk to generate role specific node neighborhoods and train node embedding exploiting the semantic of the directed graph. Also, the alternating random strategy is based on SALSA [41], which is a variant of HITS [12] and is capable to identify hubs and authorities in the graph. Finally, to evaluate the robustness, accuracy, and generalizability of the approach on several real-world datasets across three different graph theory tasks i.e. link prediction, node classification, and graph reconstruction. Also, the space and time complexities of NERD are linear, which makes it satiable for large-scale directed graphs.

Wu et al. [42] also address the issue of not incorporating the node attributes in the graph embedding vectors and proposed a scalable algorithm to extract graph embedding which also incorporates the note and edge attribute information called Scalable Attributed Graph Embeddings (SAGE). SAGE is inspired by the kernel learning framework [43] and is a unified framework for node and edge attributed graphs by utilizing adjoint graphs. This work is the first one to consider both the node and edge attributes. The computation of this novel graph embedding scales linearly with both the number of graphs and the embedding size. [42]

Li et al. [44] proposed a link prediction framework in a directed network. Specifically, they tried to analyze the role of reciprocal links information of directed closure triad and how the reciprocal links play the role in improving link prediction accuracy. They conducted the evaluation on four different types of networks which include social networks, information networks, infrastructure networks, and biological networks. Subsequently, they proposed a two weighing mechanism for link prediction by utilizing reciprocity as extra information. They evaluated their approach using the eight different directed networks such Adolescent health. [10], High-school [11], and US airports [12].

[10] https://addhealth.cpc.unc.edu/
[11] https://networks.skewed.de/net/highschool
[12] https://west.uni-koblenz.de/konect/networks/opsahl-usairport

3. THEORETICAL BACKGROUND

This chapter discusses the required technical background to understand this work. Primarily this chapter discusses the following concepts (1) Network Embedding (2) Supervised Classification.

3.1. Network embedding

Graphs such as social network graphs and graphs, in general, contain different structural patterns along with a different set of information. Analysis performed on such graphs can uncover different insights about the graph. Modeling the interaction between entities as a graph has enabled the researcher to analyze various network systems based problems in a systematic manner [45]. For instance, social networks have been used in various applications of the graphs such as friendships recommendation [46], sentiment analysis [47], and influence measure [48].

Recently, methods based on representing the network to the vector space have been quite popular. Network embedding can be broadly categorized into two categories: (1) Attributed Node Embedding (2) Non-attributed Node Embedding. Attributed node embedding vector representation incorporates the structural as well as attribute properties of the network. The ability to incorporate nodes' attributes and graph structure makes the attributed network embedding much more effective. Further, these embedding vectors can be utilized to perform downstream tasks such as node classification, link prediction, and clustering. Most of the data sets available out there let it be image, text, or number, one can argue we can classify them just by measuring the similarity between them. Here, embedding vectors come into play. For instance, in graphs, similar nodes will have node embedding vectors, when they are plotted on vector space. Similarly, embedding vectors of dissimilar nodes are farther apart from each other when they are plotted in the vector space. For this research work, we have used the following two attributed node embedding approaches (1) Attributed Social Network Embedding (ASNE) (2) Accelerated Attributed Network Embedding (AANE). The reason for choosing these two embedding extraction methods is that they primarily satisfy the following requirements. First, they have the ability to handle arbitrary types of edges (e.g., undirected or directed, unweighted or weighted) and node attributes (e.g., discrete or continuous values). Second, they well preserve the node proximity both in-network and attribute space.

Third, they are scalable to large real-world networks since real-world networks could have a large number of nodes, and respective node attributes.

3.1.1. Attributed social network embedding

Attributed Social Network Embedding (ASNE) preserves both the structural proximity and attributed proximity of the social network. To achieve this, ASNE adopted graph neural networks (GNN) to leverage the advantage of strong representation and generalization ability of deep learning. ASNE primarily considers the following proximities: (1) Structural Proximity (2) Attributed Proximity. **Structural Proximity:** Denotes the proximity of nodes that is evidenced by links/edges. For user u_i and u_j, if there exists a link e_{ij} between them, it indicates the direct proximity; on the other hand, if u_j is within the context (somehow path exists) of u_i, it indicates the indirect proximity. **Attribute Proximity:** The intersection of attribute vectors A_i and A_j indicates the attribute proximity of u_i and u_j. By enforcing the constraint of attribute proximity, we can model the attribute homophily effect, as social actors with similar attributes will be placed close to each other in the embedding space.

3.1.1.1. Structure modeling

Nodes are represented as one-hot representation, where node u_i is represented as an M-dimensional one-hot encode vector where i^{th} element of the vector is 1. The key to structure modeling is in the estimation of the pairwise proximity of nodes which is done using the softmax activation function. Let's assume there are two nodes u_j and u_i, softmax activation function is defined as the conditional probability of node u_j on node u_i, it measures the likelihood that node u_j is connected with u_i. Equations 3.1 shows the softmax activation function.

$$p(u_j|u_i) = \frac{exp(f(u_i, u_j))}{\sum_{(j'=1)}^{M} exp(f(u_i, u_{j'}))} \tag{3.1}$$

3.1.1.2. Encoding attributes

Discrete attributes are converted to numerical vectors using the one-hot encoding vectorization. Continuous attributes naturally exist on social networks, e.g., raw features of images and audios, and they are converted to real-valued vectors. For instance, textual documents are converted using bag-of-words, Doc2Vec, TF-IDF approaches. Figure 3.1 shows the example of how ASNE encodes attributes information.

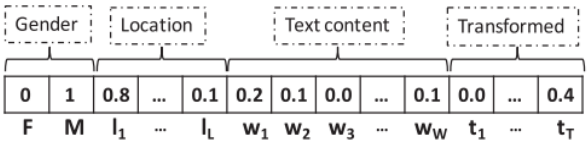

Figure 3.1. Example to show how ASNE encodes the attributes.

3.1.1.3. NN layer structure

Structure and attribute encoded vectors are further fed to the GNN. GNN is trained using the following layer structure, figure 3.2 shows the layers structure of neural network used in ASNE:

- **Input layer:** To integrate the structure and attributes, early fusion [49] is done at the input layer i.e. structure and attribute encoding vector are concatenated together at the input layer.
- **Embedding Layer:** The embedding layer consists of two fully connected components. One component projects the one-hot user ID vector to a dense vector u which captures the structural information of a node. The other component encodes the generic feature vector and generates a compact vector u_0 which aggregates attributes information.
- **Hidden Layers:** Further, u and $u^{'}$ embedding vectors are fed into the hidden layers. Hidden layers are stacked as per the tower structure, where the size of each of the successive layer is halved from the previous layer.
- **Output Layer:** Lastly, the output vector of the last hidden layer is transformed into a probability vector o, which contains the predictive link probability of u_{i} to all the nodes in U. Here, the output layer has 'Softmax' as the activation function. Also, the model is trained using the Adaptive Moment Estimation (Adam) optimizer [50].

3.1.2. Accelerated attributed network embedding

Accelerated Attributed Network Embedding (AANE) method is one of the two methods we have used to extract the attributed node embedding vectors. AANE is an efficient and scalable attributed node embedding extraction method that can learn the embedding vector representations by incorporating node attributes and network structural proximity. Also, real-world networks often contain a large number of nodes and their respective features, which demand a scalable embedding

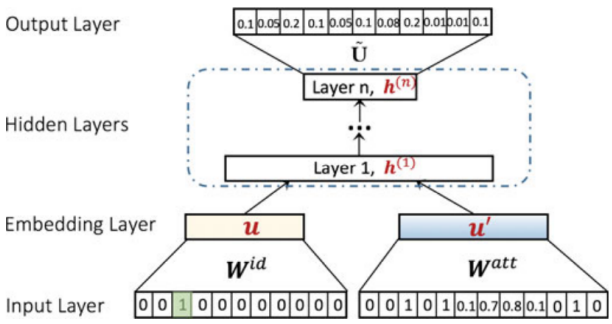

Figure 3.2. ASNE GNN layers structure.

extraction algorithm. AANE addressed the aforementioned issue by providing an optimization algorithm that operates in a distributed manner by enabling AANE to process each node by decomposing the complex modeling into many sub-problem with low complexity.

Given a network with n nodes, AANE decomposes the attribute affinity matrix S into the product of H and $H^{\intercal}$. It also imposed an edge-based penalty such that the connected nodes are close to each other in H, and the closeness is controlled by the edge weight in W. The idea is to make similar nodes in the network close to each other in H. [34]. To optimize the proposed algorithm, AANE has used the distributed algorithm such that n problems are broken down into $2 * n$ sub-problems. Different phases of AANE are discussed as follows:

3.1.2.1. Network structure modeling

To preserve the node proximity in graph G, the proposed framework is based on two hypotheses. First, a graph-based mapping is smooth especially in the region of high-density [51]. Second, the nodes with similar topology of connected nodes with higher weights are likely to have a similar embedding vector. These goals are achieved using the loss function shown in equation 3.2.

$$J_{\mathrm{G}} = \sum_{(i,j)\varepsilon\epsilon} w_{\mathrm{ij}} ||h_{\mathrm{j}} - h_{\mathrm{i}}||_2 \tag{3.2}$$

Here, h_{i} and h_{j} are the embedding vector of node i and node j respectively. $\mathrm{w_{ij}}$ is the edge weight between two node i and node j. The goal is to minimize $w_{\mathrm{ij}}||h_{\mathrm{j}} - h_{\mathrm{i}}||_2$. Larger the weight w, smaller the difference between $\mathrm{h_i}$ and $\mathrm{h_j}$ [34]. Here, l_1-norm is used as the difference metric to avoid the negative impact resulted from missing data and outliers.

3.1.2.2. Attribute proximity modeling

Given AANE incorporates the topological structure and attributes proximity into the embedding vector. Motivated by the symmetric matrix factorization [52], they use the product of H and $H^\intercal$ to approximate the affinity matrix S. The idea is to enforce the dot product of the vector representations h_i and h_i to be the same as the attribute similarity s_ij. Equations 3.4 shows the loss function to model the attribute proximity.

$$J_\mathrm{A} = ||S - HH^\intercal||^2{}_\mathrm{F} = \sum_{(i=1)} \sum_{(j=1)} (s_\mathrm{ij} - h_\mathrm{ij}{}^\intercal)^2 \tag{3.3}$$

Here, S is the affinity matrix which is calculated using the similarity matrix. Specifically, they have used cosine similarity to compute the similarity.

3.1.2.3. Combined embedding represented

To model the node proximity along with the structural proximity, both loss functions presented in equation 3.4 and 3.2 are modeled in a unified optimization problem. Following is the equation of the combined embedding representation.

$$\min{}_\mathrm{H} J = ||S - HH^\intercal||^2{}_\mathrm{F} + \lambda \sum_{(i,j)\varepsilon\epsilon} w_\mathrm{ij} ||h_\mathrm{i} - h_\mathrm{i}||_2 \tag{3.4}$$

Here λ is the constant that approximates the trade-off between the contributions of network structure and attributes information. If the value of λ is close to 0, the H will not be affected by the network structure. It is also the regularization parameter that balances the number of clusters. Each node can be a single isolated cluster if the value λ is close to 0. However, if the value of λ is significantly higher the optimal solution will end up with similar representations for all nodes, which forms a single cluster [34].

3.2. Supervised learning

Supervised learning is a subcategory of machine learning (ML), where the model is trained on the labeled input data set in order to build a predictive modeling capacity. Supervised learning can further be categorized into two broad categories i.e. (1) Classification (2) Regression. Classification is primarily for the categorical or discrete output such as if a certain image is of a car or bike. On the other hand, regression predicts the continuous output such as weather forecast, house prices.

Like any other ML model, supervised learning has two phases i.e. training phase and the testing phase. The data set is divided into two parts i.e. testing data set and training data set. Further. training data set is fed to the model in the training phase in which the model learns the patterns in the data set. Once the training phase is done, now testing phase does its work where the model is evaluated on the test data as per different available metrics. For this research work, we have used the classification supervised learning method i.e. K-Nearest Neighbour (KNN) Classifier. Following are some of the widely used classification metrics:

- **Accuracy:** is the fraction of the predictions model got right? Formally, accuracy is defined using the following formula. Accuracy is a percentage value that ranges from 0 to 100. Higher the accuracy value the better:

$$Accuracy = \frac{TruePositive + TrueNegative}{FalsePositive + FalseNegative + TruePositive + TrueNegative}$$

- **Precision:** is the ratio of true positives to the sum of true positives and false positives. A precision value closer to 1 is the most desirable.

$$Precision = \frac{TruePositive}{TruePositive + FalsePositve}$$

- **Recall:** is a measure of how well we were able to identify relevant instances. It is defined as the ratio of true positives to the sum of true positives and false positives. A recall value of 1 indicates the best performance.

$$Recall = \frac{TruePositive}{TruePositive + FalseNegative}$$

- **F1-Score:** is defined as the harmonic mean of precision and recall. F-1 score closer to 1 is the most preferable.

$$F1 = 2 * \frac{precision * recall}{precision + recall}$$

3.2.1. K-nearest neighbour

K-Nearest Neighbour (KNN) is one of the widely used supervised learning algorithms. It is an instance-based lazy learning classifier that classifies objects based on closeness. The closeness is determined by using a distance metric such as Cosine similarity, Manhattan-distance. KNN is one of the simplest classification algorithms as it does not require any prior knowledge about the distribution of the data [4]. Classification using the KNN classifier has the following stages:

1. The distance metric between each test instance and training instance is calculated. One popular distance metric which is widely used is Euclidean distance [53].

2. Based on distance metric, pick the k nearest instances to a given instance i. Then the most frequent class label among the k nearest instances is assigned to the instance i.

3. If $k = 1$ then the instance i is simply assigned to the class of its nearest neighbor.

The choice of k is critical here and it is primarily dependent on the candidate dataset. There is no such method to find the optimal value of k. Generally, a greater K value leads to stable and smooth decision boundaries. one good way to find the k is to plot the error rate between a range of k values and pick the k with minimum error rate.

4. METHODOLOGY

This section discusses the methodology used in the proposed framework: We first collected the GitHub followers data. Then, we constructed an interaction graph i.e. follow-graph to encode the 'follow' relationship in a network. Furthermore, the follow-graph is fed to the embedding extraction method that maps each node into a fixed-length vector representation. These representations are further fed to the K-Nearest Neighbour (KNN) classifier to recommend Whom to follow. Figure 4.1 shows an overview of our recommendation framework. We briefly highlighted the different phases of our framework in this section and explained each phase's details in the subsequent sections.

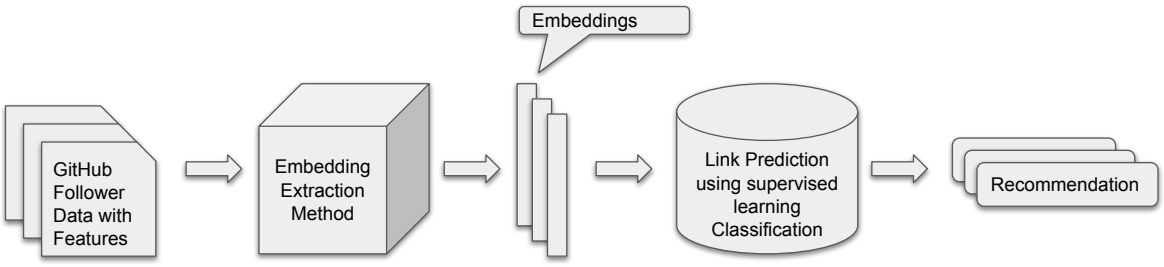

Figure 4.1. Flow diagram depicting the high level illustration of our proposed framework.

4.1. Data collection

For this research work, we have used GitHub as our data source. Specifically, we have used GHTorrent [54] to fetch the GitHub users dataset. GHTorrent maintains a data dump of GitHub event streams dataset on BigQuery [1], a Google Cloud data warehouse. GHTorrent Dataset on BigQuery contains a frequently updated GitHub dataset about various metrics, including but not limited to repository level data such as the programming language of the repository, and developer followers. We gathered GitHub user-follower data of developers who code in three popular programming languages i.e. C++, Java, and Python to conduct our experiments. Here, we consider the primary language of the developer. The primary programming language of the developers is the one, in which the developer has the largest number of source code files. Figure 4.2 shows the

[1] `https://cloud.google.com/bigquery`

distribution of the number of developers across different programming languages. Further, we filter the users who do not follow any user in the GitHub within the GitHub community. In total, we analyzed 815,420 developers. We used the GHTorrent dump until September 10, 2016. Figure 4.3 shows the database schema of GHTorrent.

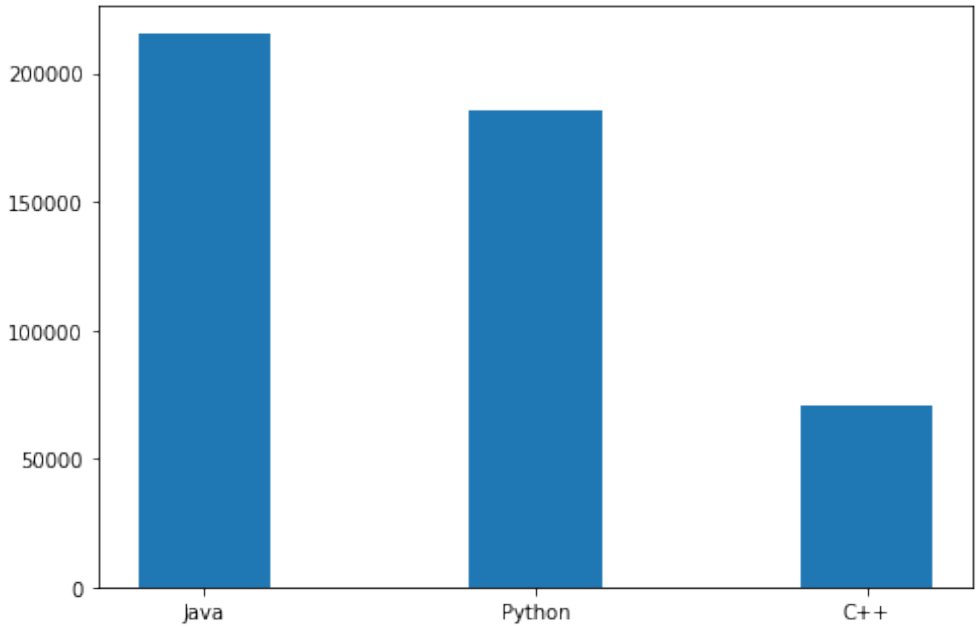

Figure 4.2. Distribution of developers across different programming languages.

4.2. Interaction graph

We created an interaction graph based on the follow behaviour of the GitHub users. We referred to this interaction graph as 'follow-graph', it reflects the developers' relationship in social activities on GitHub. In the follow-graph, the nodes represent the GitHub developers, and the edges between them represent the following relation between the connected nodes. Here, a node can have multiple attributes associated with them. Formally, follow-graph can be defined as a directed graph $G = (V, E, A)$, where V represents the set of vertices, E represents the set of edges, A represents the set of attributes associated with the vertex. If there are two GitHub users u and v, there is a directed edge between u and v if the user u is followed by v. For a user u, the number of edges going out of the u is called out-degree and the number of edges incedenting on it is called in-degree. And the degree is defined as the sum of in-degree and out-degree. Figure 4.4 shows the GitHub

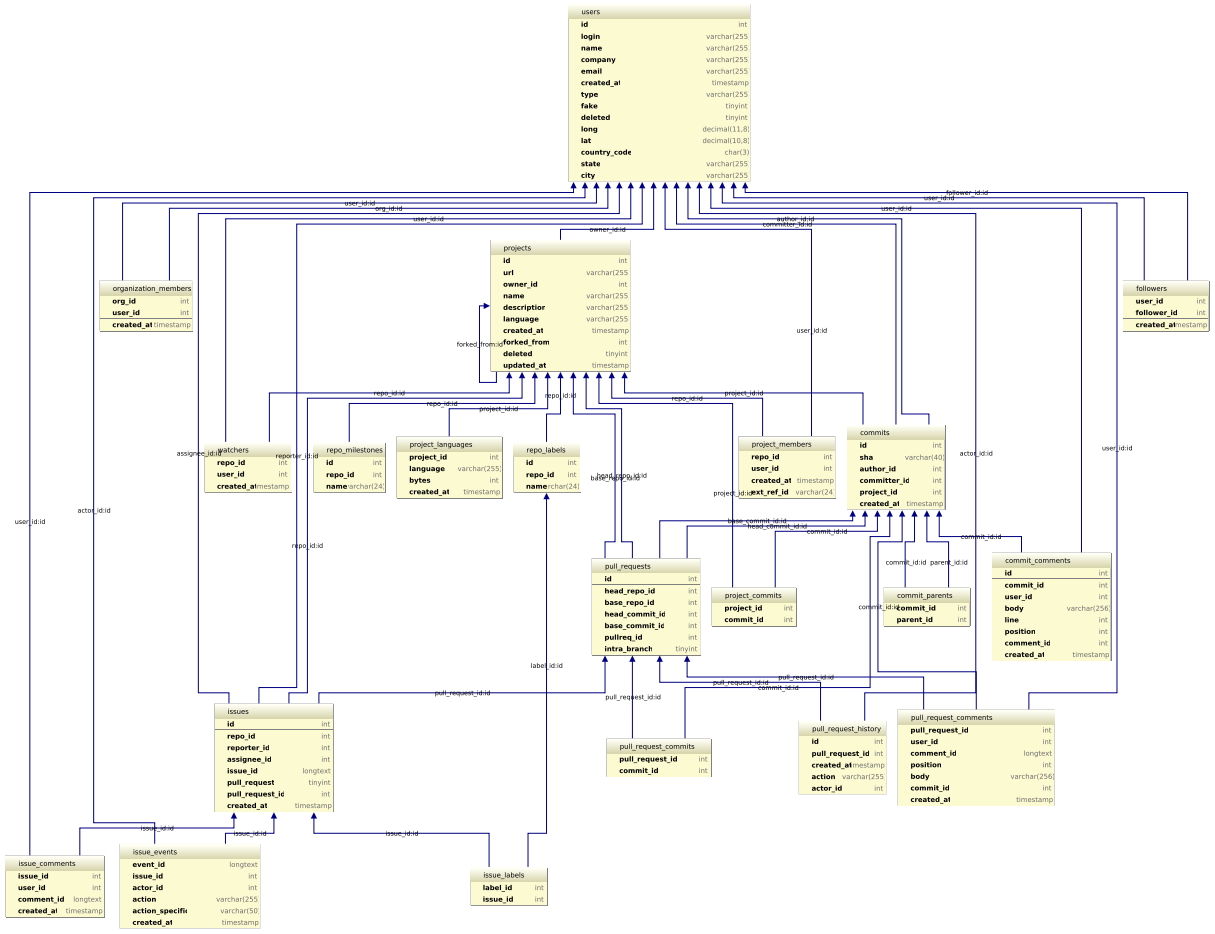

Figure 4.3. Database schema for GHTorrent.

interaction follow-graph and there is a directed edge between node u and node i. In-degree of u is 1 while the out-degree of the u is 3.

4.3. Feature engineering

Follow behavior can be associated with a different set of features, where each GitHub user has a set of associated features such as location, profile biography, and work interests. We used GitHub data REST API [2] and GHTorrent to fetch the GitHub related attributes. Figure 4.5 shows the JSON return object from GitHub API. To categorize the following behavior of the GitHub user, we categorize the user-centric features into three feature families i.e. behavior, similarity, and network each of the feature families is described as follows.

[2] `https://docs.github.com/en/rest`

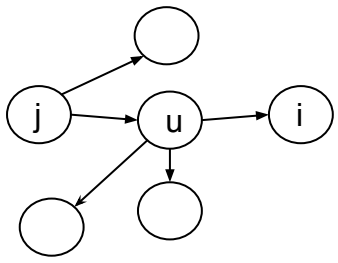

Figure 4.4. GitHub interaction graph.

- **Behavioral:** Developers follow other developers based on the observed interaction between each other. For instance, a person who may have collaborated on the projector may have replied to the question asked in the discussion forum. Hence, based on this nature of interaction a recommendation could be generated. Behavioral based features could be the following:

    1. **No. of collaborations:** Number of projects on which two users collaborated.
    2. **Number of reciprocal relationships:** Number of users who have followed back the given user.
    3. **Number of pull requests:** Pull request is a way to tell other developer about the changes you have pushed to the branch of the GitHub repository, so number of pull request could be a good feature to be included in the recommendation.
    4. **Account age:** Number of years a GitHub account has been created.

- **Similarity (homophily):** Developers tend to follow other developers with similar characteristics and interests. Following are the features that are included in our evaluation:

    1. **Programming language:** The programming language used by the GitHub user. There might be the case that developers follow other developers who use the same programming language. Also, it is possible that a GitHub user might code in multiple programming languages. Here, we only considered the primary programming language of the developer. The primary programming language of the user is the one that has the most number of source code files associated.
    2. **Hireable:** The GitHub Jobs board is a great way to find employment. This feature lets the community know if the GitHub user is available for hire. This shows if the user is

available for hire or not. here, 1 represents the user is available for hire, and 0 represents the user is not available for hire.

3. **Site admin:** A boolean feature value, where 'True' and 'False' shows if the user is the site admin or not respectively.
4. **Blog:** The feature represents the blog URL of the GitHub user. Here, we converted this feature to a categorical variable where '1' shows that the blog URL exists and '0' shows the blog URL does not exist.
5. **General information:** GitHub has a number of information stored about the GitHub user such as Profile Bio, Company Name, Location. Such textual information can be of importance in making follower recommendations. We can use the word vector approaches such as TF-IDF vectors or character level vectors. to incorporate this textual information into our node embedding vectors.
6. **Follower, Following count:** The number of GitHub followers and followees w.r.t to each GitHub user. The rationale behind including the follower and following count is that the greater the number of followers counts greater the influence would be.
7. **Number of repositories:** More repositories imply that the user has worked on more projects, and thus this feature can be a good way to measure the expertise of the user. The number of public repositories contributed to by a developer.
8. **GhGists:** This feature indicates the number of public Gists shared by a user in GitHub. Gists in GitHub are a way for developers to share useful code snippets or scripts.
9. **Account type:** This feature indicates the type of GitHub Account. GitHub accounts can be of various types, such as individual accounts or those of organizations.
10. **Number of watched repository:** This feature shows the level of interest in a developer's work. The greater the number of watched repositories, the greater the popularity of the code would be.

- **Network:** Network features are based on nodes with similar network structures. Primarily, it is calculated by the embedding extraction method. Also, we are some of the features that could be considered are Centrality, Clustering Coefficient.

Engineering transformation: Given we have three different data types of features i.e. numerical, categorical, and textual features. Before feeding these features to the embedding extraction methods, we applied some transformations to these features. We keep the numerical feature as it is. For categorical features, we converted them to one-hot encoded vectors. One hot encoded vector is essentially a binary vector, if the data point belongs to the i^{th} category then the i^{th} index of the vector is assigned a value of 1 and rest of the vector is assigned the value of 0. We encoded the textual features to a fixed-length vector of 20 dimensions using the Doc2vec [20] method. Here, the vectors are created such that the similar text would have a similar vector when they are mapped on the distance plane. In this study, we only considered the similarity and network based features that are being calculated using the embedding extraction method.

4.4. Network embedding

The ability to incorporate the node's attributes makes the network embedding much more effective. In real-world networks, nodes have rich information associated with them. For instance, in social networks there exists rich information about the social actors in addition to the link structure. Such auxiliary information can be termed as attributes. For instance, consider a Facebook friends network, where a node represents a Facebook user and the edge between two nodes represents the friendship relationship. In this type of network, the nodes have the number of information associated with it such as age, birthplace, workplace name, and location, etc. Such information plays an important role in the recommendation. In contrast to conventional node embedding approaches (includes structural information), attributed node embedding [38] has become increasingly popular due to its ability to incorporate node attribute information along with structural information. Formally, attributed network embedding can be defined as follows:

Given a set of n nodes connected by a graph G associated with edge weights W and node attributes A, we aim to represent each node $i\epsilon\ V$ as a d-dimensional vector h_i, such that the embedding representation H can preserve the node proximity both in terms of a topological structure G and node attributes A. [34].

We fed the attributed follow-graph to our embedding extraction methods i.e. Accelerated Attributed Network Embedding (AANE) [34], Attributed Social Network Embedding (ASNE) [38]. The choice of these embedding extraction methods is primarily motivated by two reasons: (1) They have the ability to handle arbitrary types of edges (e.g., undirected, directed, unweighted, and

```
{
  "login": "defunkt",
  "id": 2,
  "node_id": "MDQ6VXNlcjI=",
  "avatar_url": "https://avatars.githubusercontent.com/u/2?v=4",
  "gravatar_id": "",
  "url": "https://api.github.com/users/defunkt",
  "html_url": "https://github.com/defunkt",
  "followers_url": "https://api.github.com/users/defunkt/followers",
  "following_url": "https://api.github.com/users/defunkt/following{/other_user}",
  "gists_url": "https://api.github.com/users/defunkt/gists{/gist_id}",
  "starred_url": "https://api.github.com/users/defunkt/starred{/owner}{/repo}",
  "subscriptions_url": "https://api.github.com/users/defunkt/subscriptions",
  "organizations_url": "https://api.github.com/users/defunkt/orgs",
  "repos_url": "https://api.github.com/users/defunkt/repos",
  "events_url": "https://api.github.com/users/defunkt/events{/privacy}",
  "received_events_url": "https://api.github.com/users/defunkt/received_events",
  "type": "User",
  "site_admin": false,
  "name": "Chris Wanstrath",
  "company": null,
  "blog": "http://chriswanstrath.com/",
  "location": null,
  "email": null,
  "hireable": null,
  "bio": "🍔",
  "twitter_username": null,
  "public_repos": 107,
  "public_gists": 273,
  "followers": 21288,
  "following": 210,
  "created_at": "2007-10-20T05:24:19Z",
  "updated_at": "2021-07-29T11:10:23Z"
}
```


Figure 4.5. GitHub api response.

weighted) and node attributes (e.g., discrete, and continuous values). (2) They well preserve the node proximity both in network and attribute space.

The embedding extraction methods aim to learn a unified embedding by incorporating node attributes as well the node topology into network embedding. The extracted network embeddings behave such that developers with similar characteristics are close to each other and developers with dissimilar characteristics are further apart from each other, when mapped in space. We extracted the embedding vectors with $d = 100$ and $lambda = 0.01$ parameters. We use the implementation[3] [4]released by the Huang et al. [34] Liao et al. [38] to extract the embedding vectors.

[3]https://github.com/xhuang31/AANE_Python
[4]https://github.com/benedekrozemberczki/ASNE

4.5. Link prediction

In the link prediction task, the goal is to predict the existence of links between the pairs of nodes given a network with a certain fraction of edges removed. We took a supervised learning approach where we use node embedding as a feature to train a binary classifier. The binary classifier then predicts whether a pair of nodes will have a link using a binary class label i.e. '0' and '1'. Here, '1' represents the link between node pairs, and '0' represents the absence of a link. We use the K-nearest neighbor (KNN) classifier as our supervised binary classifier along with the Manhattan distance to find the nearest embedding vector. Literature shows that Manhattan-distance has exceptional performance as compared with other distance metrics in high dimensional spaces [55]. The reason for utilizing KNN classifier is its lazy-learning capability, which makes our framework much more efficient than using other complex machine learning models that train to learn a discriminative function using the training dataset. Lazy learning capability allows us to add more authors without re-training any classifier from scratch. Because it is an information retrieval problem, the complexity is drastically lower than traditional classifiers. As we discussed earlier in this section, each embedding vector is usually related to a single node. We have to perform some aggregation operation to convert the embedding vectors to a meaningful candidate for link prediction. In other words, we have to convert the embedding vectors of node pairs to edge embedding vectors.

Edge embedding: The challenge is to somehow combine the embedding vectors of each node in a node pair to a single embedding representation vector. In other words, we have to convert the individual node embedding vectors into edge embedding vectors. In undirected graph, this can be achieved by simply using a pair-wise aggregation operator such as summation ($u + v$), average ($u + v/2$) two participating embedding vectors [56]. However, in directed graph aggregation won't work, as we have to consider the order of the aggregating nodes which cannot be achieved using pair-wise aggregation operators. Given our interaction graph i.e. follow-graph is a directed graph we concatenated both the embedding vectors together to convert them to edge embedding vectors to ensure the node pair order.

Negative sampling: We only have the node pairs between which the link exists. To frame the link prediction task as a supervised classification problem, we also have to add the node pair between which the edge does not exist i.e. negative sampling. A major part of this framework is

negative samples. The process to add negative samples is called negative sampling. For negative sampling, we created an adjacency matrix to find the pair of nodes between which the edge exists and the edge does not exist. The existence of an edge between nodes is denoted by the values in the matrix. Here, '1' represents an edge between the node pair and '0' means represents the non-edge state between the node pair. Our data is highly imbalanced because the number of followers is quite higher than the number of non-followers. Class imbalance can be a severely complicated problem in link prediction problems and in classification problems in general. To mitigate the class imbalance problem, we randomly sampled the same of non-followers as followers. We randomly selected the node pairs between which the edges do not exist. Here, we selected the same number of non-links as the number of links between each pair of nodes.

5. EVALUATION

This chapter discusses the evaluation process and results to evaluate our GitHub follower recommendation framework proposed in chapter 4 under the number of different scenarios. Specifically, we answered the research questions discussed previously in Chapter 1. We performed our experiment on a Core i5 desktop with 16 GB RAM and for the network embedding extraction, we have used a computer cluster with 400 GB of memory.

5.1. Evaluation metrics

This section discusses the evaluation metrics we used to evaluate our framework. We primarily used four evaluation metrics: (1) Accuracy (2) Precision (3) Recall (4) F1-Score.

- **Accuracy** is one of the most popular evaluation metrics of machine learning. Accuracy is defined as the fraction of predictions our model got right. It works well when the number of samples belonging to each class is equal.

$$Accuracy = \frac{TruePositive + TrueNegative}{TruePositive + TrueNegative + FalsePositive + FalseNegative} \tag{5.1}$$

- **Precision** is defined as the ratio of true positives to the sum of true positives and false positives. Precision evaluates how precise a model performs in predicting positive labels. The precision score ranges between 0 and 1, where 1 is the best and 0 is the worst.

$$Precision = \frac{TruePositive}{TruePositive + FalsePositive} \tag{5.2}$$

- **Recall** is defined as the proportion of true positives over the sum of true positives and false negatives. Recall evaluates the number of correct positive predictions made out of all positive samples. Recall score ranges between 0 and 1, where 1 is the best and 0 is the worst.

$$Recall = \frac{TruePositive}{TruePositive + FalseNegative} \tag{5.3}$$

- **F1-score** is defined as the harmonic mean of precision and recall. Here, 'Precision' evaluates how precise a model is in predicting positive labels. And 'Recall' evaluates the number of correct positive predictions made out of all positive predictions. F1-score ranges between 0 and 1, where 1 is the best and 0 is the worst.

$$F1 - Score = \frac{2 * Precision * Recall}{Precision + Recall} \tag{5.4}$$

$$= \frac{2 * TruePositive}{2 * TruePositive + FalsePositve + FalseNegative} \tag{5.5}$$

5.2. Research questions

The section answer the research questions using our recommendation framework. Following are the research questions we answered in this section:

1. RQ 1: Can we predict whom to follow on GitHub by incorporating the user centered features such as similarity based features along with the network structural features?

2. RQ 2: How do our framework perform on the developer with low interaction?

5.2.1. RQ 1: Can we predict whom to follow on GitHub by incorporating the user centered features such as similarity based features along with the network structural features?

This research question primarily evaluates the effectiveness of our framework with the incorporation of the features discussed in section 4.3. We first extracted the embedding vectors using the GitHub followers data. Further, we fed the extracted embedding vectors to the KNN classifier. Here, we framed the problem as a binary classification problem where the node embedding vectors are used as features to train the classifier. While the class label predicts whether a pair of nodes will have a link using a binary label i.e. '0' or '1'. We have performed our experiments under the following settings (1) Attributed Embedding Setting (2) Non-attributed Embedding Setting. Under the attributed embedding setting we conducted our experiments using attributed embedding vectors which incorporate the network topology and node features. Under the non-attributed embedding setting, we conducted our experiments with an embedding vector that only considers the

node topology in the network. Table 5.1 and Table 5.2 shows the evaluation results of our framework with non-attributed and attributed embedding respectively extracted using both methods i.e. AANE and ASNE.

AANE: Under non-attributed embedding vector setting, we were able to get an F1-score of 65% and 67% for classes 0 and 1 respectively. Likewise, we were able to get precision and recall of 68% and 65% respectively for class 1. Here, we are more interested in the results for class 1, as it denotes that the edge exists between these pair of nodes, so the higher the evaluation metrics of class 1 better our model would be. For the attributed embedding setting, we were able to achieve an F1-score of 70% and 72% for classes 0 and 1 respectively. Here, we see 5% (on average) improvement in the evaluation metrics of class 1 on comparing the results of non-attributed and attributed embedding vector settings.

ASNE: We were able to get almost similar results as with the embedding vectors extracted using the AANE method. Under non-attributed embedding vector setting, we were able to get an F1-score of 63% and 66% for class 0 and 1 respectively. Also, in the attributed embedding setting we were able to achieve an F1-score of 68% and 69% for classes 0 and 1 respectively. We see a similar trend of increased evaluation metrics when we compare the attributed setting results with the non-attributed setting results. On average, we see a 3% increase in the evaluation metrics.

The difference between the results of attributed and non-attributed setting indicate that incorporating the node centric features along with the network structural features in the embedding vector results in considerable improvement. Hence, incorporating features gives us better results than the results by only considering the structural properties of the network. However, we believe the results under the attributed embedding setting are dependent on the type of features extracted with the node. For now, we have used the features discussed in the section 4.3. We believe rigorous feature engineering step is of utmost importance here. The better the features that describe the GitHub user the better recommendations would be generated. For instance, with the inclusion of 'profile bio' alone as Doc2Vec vectors, we see an increase of 2% improvement in the F1-score. In the future, we plan to incorporate features that also describe the collaboration between users such as pull requests and the number of projects shared by each user.

Table 5.1. RQ1: Non-Attributed Node Embedding Vector Results

Method	Class	Precision	Recall	F1-Score
AANE	0	64%	67%	65%
	1	68%	65%	67%
ASNE	0	64%	63%	63%
	1	65%	66%	66%

Table 5.2. RQ1: Attributed Node Embedding Vector Results

Method	Class	Precision	Recall	F1-Score
AANE	0	68%	72%	70%
	1	73%	70%	72%
ASNE	0	65%	66%	68%
	1	67%	67%	69%

5.2.2. RQ 2: How does our approach perform on the developer with low interaction?

This research question evaluates the effectiveness of our framework on the developers who do not interact often with other developers. We refer to such settings as low interaction setting. Please note that we have only considered follow as our form of interaction on GitHub. To evaluate our framework under the low interaction setting, we only selected developers with less than 10 followers in the test set of our classification model. We have used a similar methodology as used to answer RQ 1 5.2.1. First, we extracted the embedding under attributed and non-attributed setting. Further, we feed the extracted embedding vectors to the classification model i.e. KNN. Here, we train the model on the dataset containing embedding vectors for both high (number of followers $>$ 10) and low interaction (number followers $<$ 10). However, to test our framework, the test set of the classification model only incorporates the developers with low interaction.

Table 5.3 and 5.4 shows the results with the non-attributed and attributed embedding vectors respectively under the low interaction setting. For attributed embedding vectors we were able to achieve 60% and 65% F1-score for class 0 and class 1. Also, the Precision and Recall

Table 5.3. RQ2: Non-Attributed Node Embedding Vector Results

Method	Class	Precision	Recall	F1-Score
AANE	0	60%	58%	64%
	1	59%	60%	65%
ASNE	0	64%	63%	63%
	1	65%	66%	66%

Table 5.4. RQ2:Attributed Node Embedding Vector Results

Method	Class	Precision	Recall	F1-Score
AANE	0	63%	58%	60%
	1	62%	67%	65%
ASNE	0	66%	65%	66%
	1	69%	66%	67%

for class 1 are 62% and 67% respectively. Here, 1 represent the connection between two pair of nodes, and 0 represents the absence of the connection. We observed a slight decrease in the results under a low-interaction setting when compared with the results in RQ 1 5.2.1 which is expected as embedding extraction would have relatively fewer data to extract features from. For instance, with the evaluation of AANE in RQ 1 5.2.1 specifically with the attributed embedding, we got 72% F1-score for class 1 in RQ 1 5.2.1 and on evaluating our approach under low interactions we get 65% which is 5% less than the results in the usual settings. Similarly, for ASNE evaluations in RQ 1 5.2.1 we get 69% F1-score for class 1 and evaluation under the low interactions we got 67% which is 2% less then the results obtained in RQ1.

Figure 5.1. Word cloud of Github profile biography.

5.3. GitHub profile bio analysis

GitHub profile biography (bio) allows the GitHub users to add some notes about themselves such as programming languages they work with, their passion, profession, location, etc. Profile bio provides you the opportunity to broadcast to the other GitHub users about the story of your work, what defines, and inspires you, etc. With this motivation, we included the GitHub profile bio as

a core feature in the attributed embedding vectors. To make the intuitive sense of including this feature in our embedding extraction process, we visualize the GitHub profile bio using the Word Cloud. Word cloud is primarily a visualization approach that shows the collection of clusters of words in different sizes. The bigger, and bolder the word appears in the word cloud, the more often it is mentioned in the given text (GitHub bio).

Figure 5.1 visualizes the frequent words used in the GitHub profile bio. We see that developers usually write the domain they work in, their profession, and their passion. For instance, the most frequent or leading words include "Software Engineer", "Developer", "University", "Data Scientist", "Machine Learning", "Coding", "Full Stack" etc. So primarily developers write the domain they work in their GitHub profile bio. Given our evaluation metrics have been improving with the incorporation of profile bio in the embedding vectors are discussed in section 4.3, we believe that the developers follow other developers whose profile's bio are similar to their bio. In other words, they follow developers within their community and such communities are primarily based on features such as passion and profession.

6. LIMITATION AND FUTURE WORK

This section discusses some of the limitations of our research work. We have also discussed potential avenues that might address these limitations.

Data collection: We will include the developers who code in programming languages other than Python, Java, C++. Currently, we are only associating developers with a single programming language i.e. primary programming language. Here, the primary programming language is the one that has the largest number of source code files associated with it. It is entirely possible that a developer works with multiple programming languages. For instance, a developer can work simultaneously on Python as well as Java projects. In the future, we also plan to include developers who code in multiple programming languages

Embedding extraction method: Currently, we are relying on the existing attributed embedding extractions methods i.e. AANE [34] and ASNE [38]. Our approach performance is tied to the performance of the embedding method used. Better the embedding generated by the embedding methods better the recommendation results would be. Also, we feel there is a lot of room available for improvement in the existing embedding extraction method. We plan to come up with our own embedding extraction method. Our embedding extraction method would be primarily based on training a deep neural network trained using the metric-learning based loss functions such as contrastive loss function [57], Triplet loss function [58], and Lifted structured loss function [59]. At the core of deep metric learning is to learn functions such that if similar samples are fed into the model, it should output embedding that are closer to each other when measured based on a distance metric. Further, the dissimilar samples would be farther apart from each other.

Edge based embedding method: Currently, we are extracting the node embedding vectors and converting them to edge embedding by simply concatenating them together as discussed in section 4.5. In the future, we plan to use edge embedding extraction methods [60, 42], so that we can avoid the additional step of converting the node embedding into edge embedding. Also, we would extend the attributed network embedding to a general case of attributed node-edge embedding, where not only the node but also edges can contain rich information.

Improved feature engineering: Performance of our proposed framework is pretty much dependent on the quality of features used. Better the features that represent the follow interaction between the user better the recommendation would be. In this work, we have used the features described in the section 4.3. In the future, we plan to incorporate more influential features such as the impact of pull requests, the number of projects both developers worked on, and the effect of node influence measure [61] on the recommendation.

Indepth follow behaviour analysis: In the future, we plan to analyze the GitHub follow-relationship behavior in general i.e. what set of actions trigger the follow relationship. Whether the developers follow other developers who work on the same programming language, on the project, or on the same domain. Also, we will explore the effect of collaborative features such as pull requests, issue resolution on the follow-relationship.

Time based testing and training dataset: Currently, we are adding negative samples to our framework by randomly sampling the same number of node pairs that do not have an edge between them as the number of node pairs that do have an edge between them. Then, we are feeding the node pairs to the classification model. In the future, we plan to incorporate time based sampling of graphs. For instance, we can take users who joined GitHub in 2015 as a testing dataset and train our model on the users who joined GitHub in 2014. In other words, we can use newly joined users as a testing set and previously joined users as the training set.

7. CONCLUSION

There are more than 50 million active GitHub users with more than 100 million publicly available repositories [1]. Such the public nature of the repositories gives us the opportunity to mine these repositories and explore different avenues on the use of social relationships. With such a scale of information dissemination, GitHub users are often bombarded with the information. Also, such information bombardment also puts them at risk of information overload. This motivates us to come up with the framework that recommends the users which are worth following. In this work, we present an attributed network embedding based framework to recommend whom to follow on GitHub. We first constructed an interaction graph i.e. a follow-graph to encode the 'follow' relationships between the GitHub users. Further, the follow-graph is fed to the attributed embedding extraction method that maps each node into the fixed-length vector representation. The attributed embedding method confines the similar developers close to each other and keeps the dissimilar developers separated from each other in the embedding space. Further, these representations are fed to the K-nearest Neighbour (kNN) classifier to recommend the followers. The primary advantage of utilizing a nearest neighbor classifier is its lazy learning capability and its ability to incorporate new samples without the need to retrain. We were able to achieve on average 70% F1-score to recommend the user who is to be followed on GitHub.

Our work has immediate benefits in the industry, academia, and open-source community. (1) **New developer:** Often new developers wonder "Who is the best developer to follow on GitHub?", "Who should I follow to learn a programming language?", "Who should I follow in order to be updated about the latest products, and projects?". Often developers follow other developers based on popularity and social influence. Our framework would help them to get whom to follow recommendations based on various features. (2) **Information dissemination:** Following the right user can greatly improve the risk of information overload which might be generated by the irrelevant followees [62]. Also, irrelevant information or information overload, in general, may possibly affect the user intention to join or remain a member of the platform, which can ultimately affect the revenue generation of the platform. (3) **Collaboration:** Often developers follow other developers

[1] `https://venturebeat.com/2018/11/08/github-passes-100-million-repositories/`

if they work in the same organization or if they work on the same technology stack or if they have the same programming interests. The strong follower network will enable the users to collaborate at more productive level. Hence, such online collaboration would enable users to discover interesting projects and repositories more quickly and let people collaborate. Such collaborations will ultimately help in improved software development processes and better software products.

REFERENCES

[1] Yue Yu, Gang Yin, Huaimin Wang, and Tao Wang. Exploring the patterns of social behavior in github. In *Proceedings of the 1st international workshop on crowd-based software development methods and technologies*, pages 31–36, 2014.

[2] Naveen Raman, Minxuan Cao, Yulia Tsvetkov, Christian Kästner, and Bogdan Vasilescu. Stress and burnout in open source: Toward finding, understanding, and mitigating unhealthy interactions. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: New Ideas and Emerging Results*, pages 57–60, 2020.

[3] Hideaki Hata, Nicole Novielli, Sebastian Baltes, Raula Gaikovina Kula, and Christoph Treude. Github discussions: An exploratory study of early adoption. *arXiv preprint arXiv:2102.05230*, 2021.

[4] Sarim Zafar, Muhammad Usman Sarwar, Saeed Salem, and Muhammad Zubair Malik. Language and obfuscation oblivious source code authorship attribution. *IEEE Access*, 8:197581–197596, 2020.

[5] Mohammed Abuhamad, Tamer AbuHmed, Aziz Mohaisen, and DaeHun Nyang. Large-scale and language-oblivious code authorship identification. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 101–114, 2018.

[6] Daniel Schall. Who to follow recommendation in large-scale online development communities. *Information and Software Technology*, 56(12):1543–1555, 2014.

[7] Michael Brzozowski and Daniel Romero. Who should i follow? recommending people in directed social networks. In *Proceedings of the International AAAI Conference on Web and Social Media*, volume 5, 2011.

[8] Bluma Gelley and Ajita John. Do i need to follow you? examining the utility of the pinterest follow mechanism. In *Proceedings of the 18th ACM Conference on Computer Supported Cooperative Work & Social Computing*, pages 1751–1762, 2015.

[9] Abhishek Sharma, Yuan Tian, Agus Sulistya, Dinusha Wijedasa, and David Lo. Recommending who to follow in the software engineering twitter space. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 27(4):1–33, 2018.

[10] Brandon Heller, Eli Marschner, Evan Rosenfeld, and Jeffrey Heer. Visualizing collaboration and influence in the open-source software community. In *Proceedings of the 8th working conference on mining software repositories*, pages 223–226, 2011.

[11] Yan Hu, Shanshan Wang, Yizhi Ren, and Kim-Kwang Raymond Choo. User influence analysis for github developer social networks. *Expert Systems with Applications*, 108:108–118, 2018.

[12] Jon M Kleinberg et al. Authoritative sources in a hyperlinked environment. In *SODA*, volume 98, pages 668–677. Citeseer, 1998.

[13] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: Bringing order to the web. Technical report, Stanford InfoLab, 1999.

[14] Jorge E Hirsch. Does the h index have predictive power? *Proceedings of the National Academy of Sciences*, 104(49):19193–19198, 2007.

[15] Donald G Saari. The optimal ranking method is the borda count. Technical report, Discussion paper, 1985.

[16] Erzheng Fu, Yingqiu Zhuang, Jianxi Zhang, Jiayun Zhang, and Yang Chen. Understanding the user interactions on github: A social network perspective. In *2021 IEEE 24th International Conference on Computer Supported Cooperative Work in Design (CSCWD)*, pages 1148–1153. IEEE, 2021.

[17] Ronald S Burt. The social structure of competition. *Networks in the knowledge economy*, 13:57–91, 2003.

[18] Tiancheng Lou and Jie Tang. Mining structural hole spanners through information diffusion in social networks. In *Proceedings of the 22nd international conference on World Wide Web*, pages 825–836, 2013.

[19] Tapajit Dey, Andrey Karnauch, and Audris Mockus. Representation of developer expertise in open source software. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, pages 995–1007. IEEE, 2021.

[20] Quoc Le and Tomas Mikolov. Distributed representations of sentences and documents. In *International conference on machine learning*, pages 1188–1196. PMLR, 2014.

[21] Yuxing Ma, Chris Bogart, Sadika Amreen, Russell Zaretzki, and Audris Mockus. World of code: an infrastructure for mining the universe of open source vcs data. In *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*, pages 143–154. IEEE, 2019.

[22] Joao Eduardo Montandon, Luciana Lourdes Silva, and Marco Tulio Valente. Identifying experts in software libraries and frameworks among github users. In *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*, pages 276–287. IEEE, 2019.

[23] Laura Dabbish, Colleen Stuart, Jason Tsay, and Jim Herbsleb. Social coding in github: transparency and collaboration in an open software repository. In *Proceedings of the ACM 2012 conference on computer supported cooperative work*, pages 1277–1286, 2012.

[24] Prajval Mohan, Pranav Narayan, and Lakshya Sharma. Egocentric analysis of github user network. In *2021 2nd International Conference for Emerging Technology (INCET)*, pages 1–7. IEEE, 2021.

[25] Kelly Blincoe, Jyoti Sheoran, Sean Goggins, Eva Petakovic, and Daniela Damian. Understanding the popular users: Following, affiliation influence and leadership on github. *Information and Software Technology*, 70:30–39, 2016.

[26] Yingqiu Zhu, Danyang Huang, Wei Xu, and Bo Zhang. Link prediction combining network structure and topic distribution in large-scale directed network. *Journal of Organizational Computing and Electronic Commerce*, 30(2):169–185, 2020.

[27] Jing Jiang, David Lo, Yun Yang, Jianfeng Li, and Li Zhang. A first look at unfollowing behavior on github. *Information and Software Technology*, 105:150–160, 2019.

[28] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 701–710, 2014.

[29] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. Line: Large-scale information network embedding. In *Proceedings of the 24th international conference on world wide web*, pages 1067–1077, 2015.

[30] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 855–864, 2016.

[31] Chenhui Deng, Zhiqiang Zhao, Yongyu Wang, Zhiru Zhang, and Zhuo Feng. Graphzoom: A multi-level spectral approach for accurate and scalable graph embedding. *arXiv preprint arXiv:1910.02370*, 2019.

[32] William L Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pages 1025–1035, 2017.

[33] Daokun Zhang, Jie Yin, Xingquan Zhu, and Chengqi Zhang. Sine: scalable incomplete network embedding. In *2018 IEEE International Conference on Data Mining (ICDM)*, pages 737–746. IEEE, 2018.

[34] Xiao Huang, Jundong Li, and Xia Hu. Accelerated attributed network embedding. In *Proceedings of the 2017 SIAM international conference on data mining*, pages 633–641. SIAM, 2017.

[35] Svante Wold, Kim Esbensen, and Paul Geladi. Principal component analysis. *Chemometrics and intelligent laboratory systems*, 2(1-3):37–52, 1987.

[36] Shenghuo Zhu, Kai Yu, Yun Chi, and Yihong Gong. Combining content and link for classification using matrix factorization. In *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 487–494, 2007.

[37] Abhishek Kumar, Piyush Rai, and Hal Daume. Co-regularized multi-view spectral clustering. *Advances in neural information processing systems*, 24:1413–1421, 2011.

[38] Lizi Liao, Xiangnan He, Hanwang Zhang, and Tat-Seng Chua. Attributed social network embedding. *IEEE Transactions on Knowledge and Data Engineering*, 30(12):2257–2270, 2018.

[39] Shirui Pan, Jia Wu, Xingquan Zhu, Chengqi Zhang, and Yang Wang. Tri-party deep network representation. *Network*, 11(9):12, 2016.

[40] Megha Khosla, Jurek Leonhardt, Wolfgang Nejdl, and Avishek Anand. Node representation learning for directed graphs. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 395–411. Springer, 2019.

[41] Ronny Lempel and Shlomo Moran. Salsa: the stochastic approach for link-structure analysis. *ACM Transactions on Information Systems (TOIS)*, 19(2):131–160, 2001.

[42] Lingfei Wu, Zhen Zhang, Arye Nehorai, Liang Zhao, Fangli Xu, and AI Squirrel Learning. Sage: Scalable attributed graph embed-dings for graph classification. In *The International Conference on Learning Representations Workshop on Representation Learning on Graphs and Manifolds*, 2019.

[43] Bo Wu, Yang Liu, Bo Lang, and Lei Huang. Dgcnn: Disordered graph convolutional neural network based on the gaussian mixture model. *Neurocomputing*, 321:346–356, 2018.

[44] Jinsong Li, Jianhua Peng, Shuxin Liu, Xinsheng Ji, Xing Li, and Xinxin Hu. Link prediction in directed networks utilizing the role of reciprocal links. *IEEE Access*, 8:28668–28680, 2020.

[45] Palash Goyal and Emilio Ferrara. Graph embedding techniques, applications, and performance: A survey. *Knowledge-Based Systems*, 151:78–94, 2018.

[46] Giuliana Carullo, Aniello Castiglione, and Alfredo De Santis. Friendship recommendations in online social networks. In *2014 International Conference on Intelligent Networking and Collaborative Systems*, pages 42–48. IEEE, 2014.

[47] Sarim Zafar, Usman Sarwar, Zafar Gilani, and Junaid Qadir. Sentiment analysis of controversial topics on pakistan's twitter user-base. In *Proceedings of the 7th Annual Symposium on Computing for Development*, pages 1–4, 2016.

[48] Eytan Bakshy, Jake M Hofman, Winter A Mason, and Duncan J Watts. Everyone's an influencer: quantifying influence on twitter. In *Proceedings of the fourth ACM international conference on Web search and data mining*, pages 65–74, 2011.

[49] Ying Shan, T Ryan Hoens, Jian Jiao, Haijing Wang, Dong Yu, and JC Mao. Deep crossing: Web-scale modeling without manually crafted combinatorial features. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 255–262, 2016.

[50] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[51] Ahmed El Alaoui, Xiang Cheng, Aaditya Ramdas, Martin J Wainwright, and Michael I Jordan. Asymptotic behavior of\ell_p-based laplacian regularization in semi-supervised learning. In *Conference on Learning Theory*, pages 879–906. PMLR, 2016.

[52] Da Kuang, Chris Ding, and Haesun Park. Symmetric nonnegative matrix factorization for graph clustering. In *Proceedings of the 2012 SIAM international conference on data mining*, pages 106–117. SIAM, 2012.

[53] Per-Erik Danielsson. Euclidean distance mapping. *Computer Graphics and image processing*, 14(3):227–248, 1980.

[54] Georgios Gousios. The ghtorrent dataset and tool suite. In *Proceedings of the 10th Working Conference on Mining Software Repositories*, MSR '13, pages 233–236, Piscataway, NJ, USA, 2013. IEEE Press.

[55] Charu C Aggarwal, Alexander Hinneburg, and Daniel A Keim. On the surprising behavior of distance metrics in high dimensional space. In *International conference on database theory*, pages 420–434. Springer, 2001.

[56] Nino Arsov and Georgina Mirceva. Network embedding: An overview. *arXiv preprint arXiv:1911.11726*, 2019.

[57] Raia Hadsell, Sumit Chopra, and Yann LeCun. Dimensionality reduction by learning an invariant mapping. In *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, volume 2, pages 1735–1742. IEEE, 2006.

[58] Florian Schroff, Dmitry Kalenichenko, and James Philbin. Facenet: A unified embedding for face recognition and clustering. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 815–823, 2015.

[59] Hyun Oh Song, Yu Xiang, Stefanie Jegelka, and Silvio Savarese. Deep metric learning via lifted structured feature embedding. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4004–4012, 2016.

[60] Changping Wang, Chaokun Wang, Zheng Wang, Xiaojun Ye, and Philip S Yu. Edge2vec: Edge-based social network embedding. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 14(4):1–24, 2020.

[61] Adithya Rao, Nemanja Spasojevic, Zhisheng Li, and Trevor DSouza. Klout score: Measuring influence across multiple social networks. In *2015 IEEE International Conference on Big Data (Big Data)*, pages 2282–2289. IEEE, 2015.

[62] Hao Wu, Vikram Sorathia, and Viktor K Prasanna. Predict whom one will follow: followee recommendation in microblogs. In *2012 International Conference on Social Informatics*, pages 260–264. IEEE, 2012.