Developer Collaboration Prediction on Github

Songlin Qing[qing0001], Yiling Chen[yilingc], Shiqi Yang [sqyang]

I. Introduction

As the world's most popular community of open source software development, Github has millions of enthusiastic developers collaborting, forking and creating new projects every day. Because of the unique way Github is designed, it allows easy collaboration for users to work on the same project at the same time. It will be extremely useful if Github can suggest potential collaborators to users with similar software development background.

In this project, we would like to analyze the commit activities from authors to projects and interactions within the author community on GitHub, and predict if two authors will collaborate in the next quarter. Data from each quarter will be mapped to a weighted author-project bipartite graph. Then we will thoroughly compute similarity metrics on two different weighted homogeneous developer graphs folded from the weighted bipartite graph. Lastly, we will build a supervised classification model and incorporate developer features both inside and outside network topology. Feature importance of different similarity metrics will be compared and analyzed.

II. Related Work

Link prediction has been a focused research area in recent years, given its wide application in academic, governmental and commercial uses. Liben and Kleinberg (2007) [1] formalized the link prediction problem and evaluated different approaches to predict link formation by measuring the proximity of nodes in several un-weighted homogeneous co-authorship networks G_{collab}. Though the methods performed much better than a random predictor, the overall accuracy was not high. When narrowing down the nodes to be analyzed, they picked nodes that appeared in both training and test sets with more than 3 degrees, which biased the estimation of prediction accuracy. The authors agreed that G_{collab} is a lossy representation of the bipartite network as much information are lost in the projection process.

To preserve the collaboration strength information, Newman (2001) [2] proposed an edge weight assignment formula that considers the collaboration project size and collaboration times. Zhu and Xia (2016) [3] carried out empirical experiments on 4 weighted homogeneous networks and showed that the weighted scores outperformed the traditional un-weighted indices.

Grover and Leskovec(2016)[11] proposed a node representation learning algorithm, node2vec, and showed that the node embedding can be used to predict missing link with better performance than using local similarity and spectral clustering. Gao et al (2018)[12] improves the random walk approach for bipartite networks to encode implicit relations.

Sa and Prudencio (2011) [5] combined supervised learning and weighted graphs and achieved a satisfactory result. However, they only incorporated the metrics computed from the network structure, not information available outside the network.

III. Data set and Graph Representation

A. Dataset

We propose to use the GHTorrent[6] dataset, which is an archive of the Github public events queried through the Github REST API as far back as 2007. The dataset contains granular information of Github users, organizations, repositories, programming languages, commits, pull requests and issues. As of Oct 2018, the GHTorrent dataset includes 46.7M repositories and 16.2M users. In our project, we queried GHTorrent via Google BigQuery.

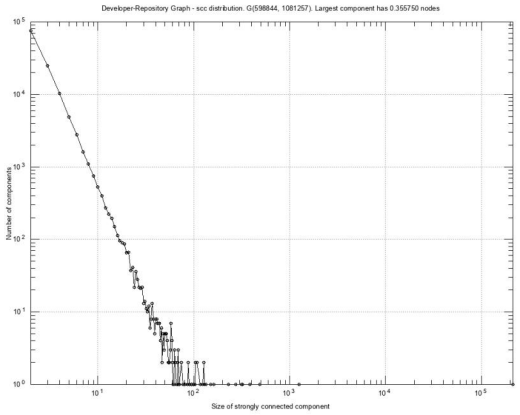
With the goal of predicting developer collaboration, we retrieve all commits in the GitHub repositories that are:

1) tagged with the Python programming language;
2) have two or more commit authors in the first quarter of 2016.

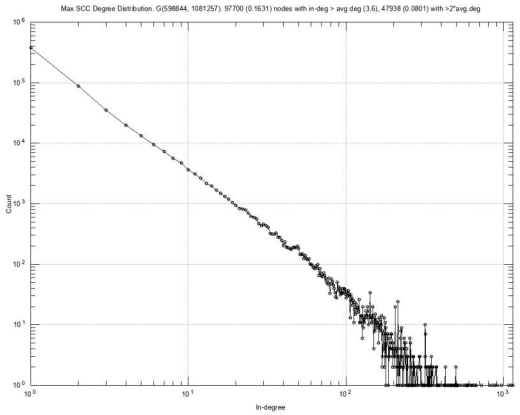
In the resulting data set, there are $326,996$ repositories, $271,848$ commit authors[1], and $1.08M$ repository author combination that has non zero commit count. We also retrieve the commits with the same criteria for the second quarter of 2016 as our validation set.

B. Graph Representation

1) Developer-Repository Graph: We can define a developer-repository bipartite graph based on the commit dataset where there are two modes of nodes - developers and repositories, where a developer and a project is linked when the developer has authored one or more commits in the project. The link can be optionally weighted by the number of commits by the author in the project over the given period of time. The resulting bipartite graph has 599K nodes and 1.08M edges, with one giant component covering 213K nodes and 796K edges. The approximate diameter of the giant component with 100 sample is 29. Some network characteristics are shown in Fig 1.



(a) Size distribution of strongly connected components



(b) Degree distribution of the giant component

Fig. 1: Network characteristics of the developer-repository graph

2) Developer Collaboration Graph: The two-mode developer-repository graph can be projected to a one-mode developer collaboration graph where two developers are connected when they have authored commits to one or more repositories. For the weighted graph, there are different ways to project the weights:

1) Newman (2001)[2] proposed an edge weight assignment formula that considers the collaboration project size and collaboration times.

$$w_{ij} = \sum_k \frac{\delta_i^k \delta_j^k}{n_k - 1} \tag{1}$$

Where n_k represents the number of authors who committed to the repository, δ_i^k is 1 if author i committed to project k and 0 if not. Each collaboration is normalized with the total authors and combining collaborations together gives us the strength of collaborations between author i and author j.
With the Newman score, authors who have committed to the same repository have an edge in the one-mode projection. However, this score does not take into account how many commits have the author performed thus two users who committed to a project once each still shows connected.

2) We also propose an alternative weight assignment for the projected edge, **Common**

[1]Note that we use commit author, the person who initially wrote the code, instead of the committer, the person who performs the commit, to better capture the collaboration. When the author differs from the committer, it is usually due to that the author does not have commit right to the repository.

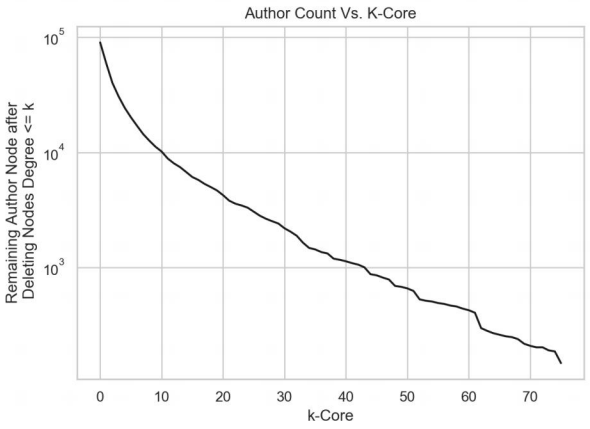


Fig. 2: K-Core Decomposition

Commit Index(CCI):

$$\texttt{CCI}_{ij} = log(\sum_k min(c_i^k, c_j^k))$$

Where c_i^k is the commit count for developer c in project k. Here we consider the collaboration between two developers to be the lesser number of commits between the two in each project summing over all projects they collaborate in.

C. Scope Narrow-down

As covered in section B, the largest Strongly Connected Component in the bipartite graph contains 213K nodes including 89.9K authors. There are 4B possible collaborating node pairs, which makes even the simplest local similarity scores calculation a non-trivial task. Liben and Kleinberg (2007)[1] filtered out authors who published less than 3 papers in both training and test periods in the G_{collab} analysis. We think this practice is flawed as the test data is leaked into training set and the resulting graph is not necessarily connected. Instead, we implemented k-core decomposition [7] to prune the author and project nodes on the bipartite graph to narrow down the number of author node pairs to a manageable size, while keeping the MxSCC connected.

As author and project node share similar degree distribution, we pruned author and project degrees at the same rate. As shown in Fig2, after removing authors with 25 or less projects and projects with 25 or less authors there are 3053 authors remaining in the graph. This pruning process result in 4.6M possible collaborating node pairs, and narrow down our analysis scope to highly active developers (more than 25 projects) and medium to large sized projects (more than 25 authors).

IV. Approach

In this section, we implement several metrics of similarity on weighted author graph folded from the author-repository bipartite graph. We analyze local/global features and community features to obtain the similarity between each node pair.

Let $\Gamma(x)$ be the set of neighbors of node x, and $w(x, y)$ be the edge weight between node x and node y. Because our graph is undirected weighted graph, $w(x, y) = w(y, x)$.

A. Local Similarities

- **Common Neighbor** (CN):

$$\sum_{z \in |\Gamma(x) \cap \Gamma(y)|} w(x, z) + w(y, z) \qquad (2)$$

The most basic approach is to count the number of common neighbors between author x and author y. A fair amount of common neighbors indicates the two authors are likely to collaborate in the future. In the weighted graph, we also took the edge weights between their common neighbors and themselves into account.

- **Jaccard's Coefficient** (JC):

$$\sum_{z \in |\Gamma(x) \cap \Gamma(y)|} \frac{w(x, z) + w(y, z)}{\sum_{a \in \Gamma(x)} w(a, x) + \sum_{b \in \Gamma(y)} w(b, y)} \qquad (3)$$

The Jaccard's coefficient values the similarity between pair of nodes with low degrees more than pair of nodes with high degrees. In our case, if two authors both are highly dedicated authors who commit more than 1000 projects within a quarter, their similarity score will be downsized by the number of collaborators on their own.

- **Adamic-Adar Coefficient** (AA):

$$\sum_{z \in |\Gamma(x) \cap \Gamma(y)|} \frac{w(x, z) + w(y, z)}{log(1 + \sum_{c \in \Gamma(z)} w(c, z))} \qquad (4)$$

The Adamic-Adar coefficient is a derivate of the JC coefficient. It assigns a higher similarity score to common neighbors with low degrees (Sa, & Prudencio ,2011). If two authors both collaborate to a highly dedicated authors with a large group of collaborators, their similarity score will be lower than neighbors with a small group of collaborators.

- **Preferential Attachment** (PA):

$$\sum_{a\in\Gamma(x)} w(a,x) * \sum_{b\in\Gamma(y)} w(b,y) \qquad (5)$$

Because the degree distribution of the user graph follows the power-law distribution, we also included Preferential Attachment as one of our proximity metrics. PA assumes that nodes with high degrees are more likely to link to a new node. In our case, the two author might collaborate in the future if they both have large weights with their existing collaborators.

B. Katz

The Katz [8] sums over the paths between two nodes with different lengths and penalize the longer paths exponentially by the value of $\beta(0 < \beta < 1)$.

$$\sum_{l=1}^{\infty} \beta^l \cdot \left| paths_{x,y}^{<l>} \right| \qquad (6)$$

The matrix of Katz scores can be calculated by $(I - \beta A)^{-1} - I$, where A is the adjacency matrix of the graph. In our analysis, we calculated Katz score for Newman, CCI and unweighted graphs and tested the accuracy with β = [0.05, 0.005, 0.0005, 0.00005]

C. Node2Vec

Node2Vec generates an embedding for the nodes in the network that maximizes the log likelihood of the network neighbourhood generated following a 2nd order random walk strategy S:

$$\max_f \sum_{u\in V} logPr(N_S(u)|f(u))$$

After we learn the node embedding with node2vec, we can generate a representation of each node pair by combining the embedding of both nodes using a binary operator. The node pair representation can then be leverage for link prediction.

D. Normalized Spectral Clustering

Define the weighted adjacency matrix (W): $W = [w_{ij}], wij = w(i,j)$, the weighted degree matrix (D): $D = [d_{ii}], dii = \sum_{j=1}^{n} w_{ij}$. According to Shi and Malik (2000)[9], the normalized spectral clustering works with eigenvalues and eigenvectors of the normalized Laplacian matrix (L_n):

$$L_n = D^{-1}L = I - D^{-1}W \qquad (7)$$

The reason we choose normalized instead of unnormalized spectral clustering is that the normalized one obtain clusters by achieving two objectives: minimize the between-cluster similarity and maximize the within-cluster similarity, while the unnormalized one only implements the first objective.

Then we select the number of clusters by finding the rank k that maximizes eigengap Δ_k : $\Delta_k = |\lambda_k - \lambda_{k-1}|$.

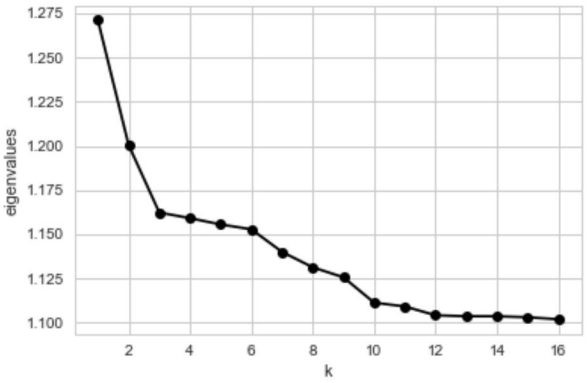


Fig. 3: First 16 eigenvalues of Newman

In the Newman graph, k = 2, so we split the nodes into two clusters corresponding to the positive and negative entries in the eigenvector of the second smallest eigenvalue. In CCI graph, k = 9, so we run the k-means algorithm on the first 9 eigenvectors to split nodes into 9 clusters. Then we transfer the clustering results to whether each pair of nodes is in the same cluster in the supervised learning model. Though we focus on link prediction, community features can be an

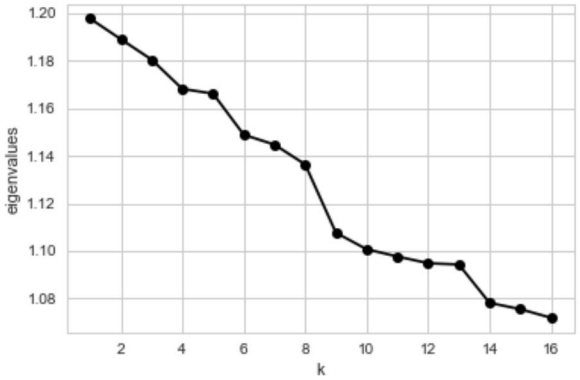


Fig. 4: First 16 eigenvalues of CCI

important indicator under supervised learning.

V. Model Evaluation and Results

A. Baseline Model

We filter data in the first quarter of 2016 as the training set and the second quarter of 2016 as the test set. A random predictor is defined as our baseline model. Unlike the model proposed by Liben et al, we include nodes that are in the training set but disappeared in the test set because we also want to predict if users would become inert in the future. Define the set of 3053 authors in the largest strongly connected component(MxScc) after K-Core decomposition in the author-repository bipartite graph as V_{Scc}, the author graph that we obtain by folding MxScc as $G_1 = (E_1, V_1)$, and the test set as $G_2 = (E_2, V_2)$. Since our primary goal is link prediction, we use the area under the Receiver Operating Characteristic (ROC) curve, or AUC[10] as the metric for evaluation, which is common in the link prediction literature.

The random predictor is defined as randomly generate a score for each node pair and we compare the score and whether the link appears in G_2 to obtain the AUC. The AUC for the random predictor is 0.50.

B. Model Performance

1) Local Similarity: We implement four local similarity algorithm on the developer graph with different weights. In each method, we compare the node pair scores with the the actual new edges to obtain an AUC score. The results are:

TABLE I: Local Similarity AUC Scores

Weight	CN	AA	JC	PA
Newman[2]	0.956	0.956	0.949	0.696
CCI[3]	0.953	0.953	0.950	0.792
Unweighted	0.949	0.949	0.949	0.783

It is observed that Preferential Attachment score performs the worst, while the other three scores have much better and similar level of prediction power. Newman score outperforms CCI slightly with Common Neighbour and Adamic-Adar, but CCI performs much better with Preferential Attachment. Un-weighted graph scores are inferior to the better of the weighted graphs with all local similarities.

2) Katz: Katz score with different β values are computed on all three types of graphs, to capture the path based importance features. The results are:

TABLE II: Katz AUC Scores

Beta	0.05	0.005	0.0005	.00005
Newman	0.409	0.957	0.956	0.956
CCI	0.460	0.347	0.318	0.961
Unweighted	0.390	0.388	0.952	0.956

It is observed that Katz scores with high β value have poor performance while low β value have better performance. This behavior shows the number of longer paths between two nodes have adverse impact on the link prediction accuracy. CCI with β = 0.00005 has the best performance out of the three graphs.

3) Node2Vec: We ran node2vec on the folded graph with the following hyper-parameters: dimensions $d = 32$, walks per node $r = 24$, walk length $l = 400$, context size $k = 10$, with varying return parameter p and in-out parameter q. After we obtain the node embedding, we follow the original node2vec paper[11] and combine the node embedding into node pair embedding then run supervised learning for link prediction. The authors listed four different binary operators for combining the node embeddings into node pair embedding.

[2]The score proposed by Newman (2001) [2] (in section III-B-2a).
[3]The common commit index proposed by us (in section III-B-2b).

We found that for our dataset the weighted L1 operator perform the best, so we fixed the binary operator to weighted L1 operator for the evaluation in table III.

TABLE III: Node2Vec AUC Scores

	p=1,q=1	p=0.1,q=10	p=10,q=0.1
CCI	0.949	0.937	0.951
Unweighted	0.943	0.944	0.947
Bipartite	0.923	0.915	0.928

Node2Vec representation with different p and q values performed equally well with CCI weighted graph. It suggests both local and global similarity are strongly predictive in link formation. In all three graphs, node2vec with the exploration strategy that encourages outward exploration ($p = 10, q = 0.1$) performs the best. Note that node2vec on the original bipartite graph performs the worst out of the three, potential it is due to the fact that with the same walk length, a random walk only reaches half of the number of authors in a bipartite graph.

C. Supervised Learning

Liben and Kleinberg (2007)[1] compared the performance of different link predictors on 5 real-world networks and concluded there is no single clear winner among the technique. In machine learning, an ensemble of models are usually effective in improving the prediction effectiveness and outperform the individual models. Besides topological information, other information of the node and interaction can also be helpful in predicting future interactions. Al Hasan, Chaoji, Salem, and Zaki (2006)[4] modeled the link prediction task as a classification problem and introduced a set of calculated features to improve the classification performance. In our supervised learning models, we use 3 non-network features, 8 local similarity scores, 3 katz scores, 2 Node2Vec scores, and 2 community scores to predict the developer collaboration. The 3 non-network features are:

1) Difference in author account age
2) If the authors are from the same country
3) If any of the authors did not make any commit in the last month of the quarter

To formulate the supervised learning problem, we identify each node pair as a data point and predict if the node pair is going to form in the following quarter. A standard train-test split was performed on the dataset and 4 classification models are trained with 5 folds of cross-validation. With the extreme imbalance in the training set (5.7% of node pairs will develop an edge), undersampling is applied and accuracy is used as an optimization metric to train each model. Each trained model is then applied to the whole training set to produce a probability score that an edge is going to form between the node pairs. All scores are compared with the actual new edges to produce an AUC score. The best model is then applied to the test data to produce an unbiased estimation of test error. Here are the results:

TABLE IV: Supervised Learning AUC Scores

Stage	Logistic	Decision Tree	Random Forest	XGBoost
Train	0.918	0.948	0.989	0.960
Test	-	-	0.972	-

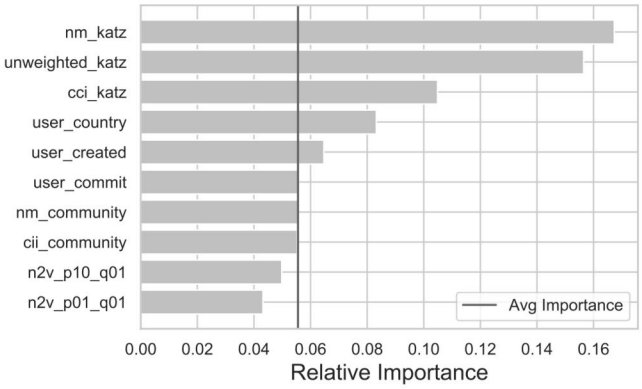


Fig. 5: Top 10 Feature Importance

Feature importance (Fig6) of the winning Random Forest model shows the most important feature as the Katz scores, followed by the non-network features, community feature, and Node2Vec similarities. The feature importance tells us the Katz score perform pretty well, especially on newman weighted and unweighted graphs. The local similarity scores did not appear on the top 10 list, possibly because there are 8 correlated local similarity scores in this feature set thus each feature getting little explanatory power.

To understand the importance of feature categories, we reduce the feature set to include only

one best performing feature from each score category:

1) Local Similarity: Adamic-Adar score on Newman weighted graph
2) Katz: $beta$=0.0005 on CCI weighted graph
3) Node2Vec: Weighted p=0.1,q=10 (Though this is not the best metric, it represents node structural similarity which is not represented by other features)
4) Community: 9 communities based on CCI weighted graph
5) Non-network features: Account age diff, Same country, March Commit

With less features, the model is not as accurate but our intention is to assess the importance of features:

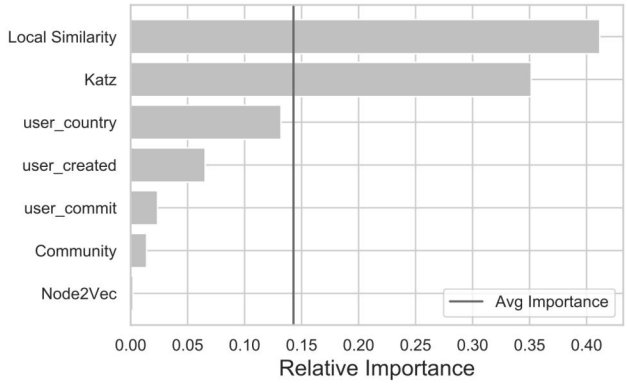


Fig. 6: Feature Importance (One Feature Per Category)

As we can see, community information adds little to the model while Node2Vec does not show any importance at all. It may be either because the author nodes that are similar in structure are not likely/unlikely to collaborate than random authors, or the similar information is explained by the other fewtures. The most important features are local similarity, Katz, and if the authors are from the same country.

VI. Future Work

Due to constraints in time and computing power, we have narrowed down the scope and only analyzed the highly active developers and medium to large sized projects. In order to evaluate our different model consistently while preserving generality, we will explore different techniques to reduce and/or distribute the computation.

Another area of exploration is alternative node pair embedding. Outside of the limited set of binary operators we tried to transform node embedding to node pair embedding, there might be better way to combine the node representation or ways to learn the node pair representation directly.

We planed to use 9 quarters of data to train our supervised learning model but stopped with 1 quarter because of limited features. If time allows, we will extract more features from both network topological information and other developer/repository features. With more features and more training data, the model performance should continue to improve.

The source code of the project can be found on: `https://github.com/leomaxx/GithubNetworkAnalysis`

References

[1] Liben-Nowell, D., & Kleinberg, J. (2007). The link prediction problem for social networks. Journal of the American Society for Information Science and Technology, 58(7), 1019-1031.

[2] Newman, M. E. (2001). Scientific collaboration networks. II. Shortest paths, weighted networks, and centrality. Physical review E, 64(1), 016132.

[3] Zhu, B., & Xia, Y. (2016). Link prediction in weighted networks: A weighted mutual information model. PloS one, 11(2), e0148265.

[4] Al Hasan, M., Chaoji, V., Salem, S., & Zaki, M. (2006, April). Link prediction using supervised learning. In SDM06: workshop on link analysis, counter-terrorism and security.

[5] Sa, H. R., & Prudencio, R. B. (2011). Supervised link prediction in weighted networks. The 2011 International Joint Conference on Neural Networks. doi:10.1109/ijcnn.2011.6033513

[6] https://bigquery.cloud.google.com/dataset/ghtorrent-bq:ght

[7] Alvarez-Hamelin, J. I., Dall'Asta, L., Barrat, A., & Vespignani, A. (2005). k-core decomposition: A tool for the visualization of large scale networks. arXiv preprint cs/0504107.

[8] Katz, L. (1953). A new status index derived from sociometric analysis. Psychometrika, 18(1), 39-43.

[9] Shi, J. and Malik, J. (2000). Normalized cuts and image segmentation. IEEE Transactions on Pattern Analysis and Machine Intelligence, 22 (8), 888 905.

[10] Mason SJ and Graham NE. (2002) Areas Beneath the Relative Operating Characteristics (roc) and Relative Operating Levels (rol) Curves: Statistical Significance and Interpretation. Quarterly Journal of the Royal Meteorological Society, 2002, pp 21452166.

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

[12] Gao, M., Chen, L., He, X., and Zhou, A. (2018). BiNE: Bipartite Network Embedding.