CS685 - DATA MINING

GitHub Reviewer Recommendation using Graph

Harshvardhan Pratap Singh - 20111410 hrshsengar20@iitk.ac.in

Yash Saraswat - 20111073 yash20@iitk.ac.in
Gaurav Tank - 20111407 gauravt20@iitk.ac.in

December 6, 2020

Abstract - For the larger projects on the platform like GitHub, Collaboration is done by cloning the repository, modifying the sources, and then issuing the pull-requests. For big scale projects, the assignment of reviewers becomes a challenging task. For example, multiple reviewers might review the same pull-requests, a reviewer may check the pull-request from out of his/her domain. That may lead to a considerable feedback loop time, and sometimes the critical changes might be ignored. So, we are building a recommendation system for recommending reviewers to the pull requests so that such a system can save reviewers' time and improve overall code quality due to a short feedback loop time. In this work we are trying to build the GitHub Reviewer Recommendation System for the Kubernetes repository on GitHub and trying to generalize it for other large projects. We will be building a Graph Neural Network and compare its performance with existing Supervised Random Walk implementations.

Index Terms - GitHub, Reviewer Recommendation System, Graph, Graph Neural Networks (GNN), Supervised Random Walk (SRW)

1 Problem statement

Being the largest open-source community hosting millions of software repositories, GitHub popularizes the pull-based development model where external contributors outside the core team can propose source code changes and resolve issues. So, the number of Pull-Requests is substantially high and can overwhelm the core team maintenance schedule, leading to late responses and critical Pull-Requests being ignored. Automated testing workflows are helpful, but before merging, each Pull-Request still needs manual and thorough reviews to enforce high engineering standards. It happens that pull request creators tend to tag other contributors or core team members to review the code changes. Also, in some cases, pull request creators do not have an idea about the reviewers or core team members; finding an appropriate reviewer takes much time and the feedback by reviewer may get delayed.

The project core team must act as a code quality checker, ensuring that the code is thoroughly inspected before merging into the main branch. However, due to many pull requests, this task becomes computationally expensive. To increase the inspection process's speed, we need that the pull request reaches to appropriate reviewer, and for that task, we need supervision. So we propose a supervised approach to recommend reviewers for an incoming pull request so that the inspection of code can take place at a faster pace. We implement Graph Neural Network and analyze the existing machine learning models for the same task.

2 Introduction

Software Development Cycle (SDC) is a challenging task for software projects. It gets difficult for huge-scale projects. The big companies use specialized tools to manage their software code-bases. However, small companies, startups, and open-source projects cannot afford to develop such tools. Many open-source Version Control Systems (VCS) are available in the market to deal with this. Github (GH) is an example of the VCS widely used in academia, startups, startups, and sometimes even in the industries. Most of the version control systems support the pull-request (PR) based method to collaborate on the project. Github popularizes the pull-based development model where external contributors outside the core team can propose source code changes and resolve issues. External contributors makes a copy of main branch code and work on them, when they create new functions or add new functionalities then the code in the base repository should be updated to do that, the contributors create a pull request to merge their work in the base repository. Pull request is a formal way, where we're proposing our changes and requesting that someone review and pull in our contribution and merge them into their branch.

Taking inspiration from literature H. Ying et al. [7] and Yue Yu et al. [8] on reviewer recommendation problem, considering both non-graph and graph based solutions for personalized recommendation. This project aims to analyse the the literature findings on a large-scale recent project such as Kubernetes. We apply a prototype of the Supervised Random Walk and analyse its features and hyperparameters.We also find the reason for its success/failure. We then apply Graph Neural Network for reviewer recommendation and analyse the findings in the process of learning a machine learning model.Both the approaches use graph based methods to recommend reviewers for pull requests.

3 Dataset

We collected the past pull-requests' data from the repository named 'Kubernetes' hosted at GitHub to construct the dataset. We considered the past pull-requests of Kubernetes repository because it is large scale project and contains adequate amount of pull-requests. GHArchive is a project, which records the public GitHub repositories. We obtained the dataset from it using Google's Big Query API. The dataset was in a gigantic tabular format consisting of approx 130k rows, each of which had total 28 columns shown in the table 1.

Column Name:
Pull Request ID:
Github Pull Request ID:
Commenter ID:
Commenter Username:
Commenter Follower Count:
Commenter Total Github Commit Count:
Commenter Base Repository Commit Count:
Head Commit Author ID:
Head Commit Author Username:
Head Commit Committer ID:
Head Commit Committer Username:
Base Commit Author ID:
Base Commit Author Username:
Base Commit Committer ID:
Base Commit Committer Username
Head Repository ID:
Base Repository ID:
Base Repository Owner:
Base Repository Owner Username:
Head Repository Owner ID:
Head Repository Owner Username:
Head Commit ID:
Base Commit ID:
Comment Created At:
Comment Position:
Comment ID:
Comment Body:

Table 1: GH Archive Dataset Schema

This data table is a snapshot of the Repository's past data between year 2015 to 2018. The table may have different rows for different comments on individual pull-requests. From this dataset, the different pull-requests, comments, and entities are collected. Now, comment body usually does not provide usual information as comments can be either a question or an answer by the reviewers and it may contain simplest comments such as 'fixed', 'done', 'added' etc.

To get the information about the individual pull-requests, For each pull-request, we scrapped the file modifications, file additions, file deletions about the pull-requests. We use Github API to achieve this.

Column Name:
Pull Request ID:
Pull Request State:
Pull Request Title:
Pull Request Creation Time:
Pull Request Updation Time:
Pull Request Closed Time:
Pull Request Merged Time:

Table 2: PR File Changes

Column Name:
Pull Request ID:
Modified Filename:
File Status:
Additions in File:
Deletions in File:
Changes in File:

Table 3: Pull Request Titles Schema

Github provides 60 requests per hour for an IP address. Though it also provides 5000 requests per hour per authorized user. Our dataset has total 11474 distinct pull requests. So, we split the it into the three batches containing 4096, 4096 and 3282 pull-requests respectively. From each batch, candidate pull-requests are fetched using python3 module named requests. To speedup this process, threading module is used. Similarly, pull-request titles are also fetched using the same method. The schema of pull-requests' titles and filepaths changes shown in the table 2 and table 3 respectively.

4 Methodology

Non-Graph-Based Approaches:

To recommend the reviewers for incoming pull-request, many approaches, both graph based and non-graph methods are established by the researchers. In this section, some of the non-graph based methods from the literature[8] are discussed. We will discuss three different methods viz. Simple (Baseline) method, Support Vector Machine (SVM) Based recommendation, Information Retrieval (IR) based recommendation.

From above mentioned methods, SVM and IR methods uses the vector space model of the pull request. To create vector space model of some incoming pull-request, we first remove gibberish characters from the pull-request's comments and titles. then it uses porter-stemmer to convert the pull-request into a feature vector. of which each element in the is a term, and the value gives the importance for the pull-request. The more a given word appears in a pull-request, the more important it is for that pull-request. TF-IDF method is used to indicate the value of a term. This way all terms can be calculated to form a vector model of the pull-request.

• Simple (Baseline) Method: In this approach, top-k reviewers who reviewed the most past pull-requests are selected. That is suggesting top-k reviewers from of the list of reviewers sorted on the number of pull-request they have reviewed.

• Support Vector Machine Based Recommendation: One pull-request can have many reviewers so, in this method, problem is modelled as a multi-class tagging problem. To solve this problem, SVM is trained for all classes (reviewers in this case) using one vs all approach on the past dataset of pull-requests and corresponding assigned reviewers. The input-set is vector space model of the pull requests and output are reviwers. Now when new candidate pull-request issued, for all reviewer candidates, probabilities are identified and then selected the top k reviewers with the most probabilities.

• IR Based Recommendation: In this approach, vector space representation of pull request acts as bag-of-word representation of pull-request is created using its title and comment content and then similarity is calculated for all pull requests reviewed by a reviewer and then cosine similarity is summed up and this quantity is called score of candidate pull-request on reviewer. This score is identified for all reviewers using the candidate pull-request. And then selected top-k reviewers with the highest score.

Graph-Based Approaches:

A lot of research focuses on traditional non-graph based approach[3] to connect reviewers with pull request. But graph based approaches showed significant improvement over them because Pull-requests contains rich features and those were not utilized

properly in non-graph based approaches. But rather non-graph based approaches could focus on a particular domain e.g. expertise, common interest. Graph based approaches can work with mixed approaches, because integrating reviewer's expertise and comment interests, represent significant improvements in precision and recall, and the overall performances of mixed approaches are more stable than using different approaches independently. Our work here is extension upon Yu et al. [7] and Ying et al. [8]. In these papers they used interaction between reviewers and PR owners. And also information from Github Profile pages. That information acted as features for reviewers. This graph based approach automatically learns reviewer's authority and expertise then recommend best reviewers for incoming PR. Graph construction and propagation our approach is combining these approaches with File location and information retrieval based approaches and kind of assigning edge weights on these basis. The network structure utilization of activity between all GitHub users is a promising direction of research because it can lead to reviewer recommendations from a larger pool of users connected by social interaction and their profile also from sematic text similarity. Given a large-scale repository. We constructed graph from interaction comment available for PR review interactions, title, description of PR as well as Github profile information of reviewers. Incoming PR finds the similarity of the Pull- Request to previous ones and then picks the top 20 among them and connect with them. Data is extracted from various sources GHTorrent [4] , GHArchive, Github API using data of large scale projects such as Kubernetes, brew, etc.

4.1 Supervised Random Walk

Supervised Random Walk is one of the widely used techniques on graph data sturcture where the dataset contains imbalanced class distribution. We have used SRW to understand the relationship between pull requests and reviewers. To achieve the above goal we have implemented a prototype of Supervised Random Walk so that we can analyse its working on our data.

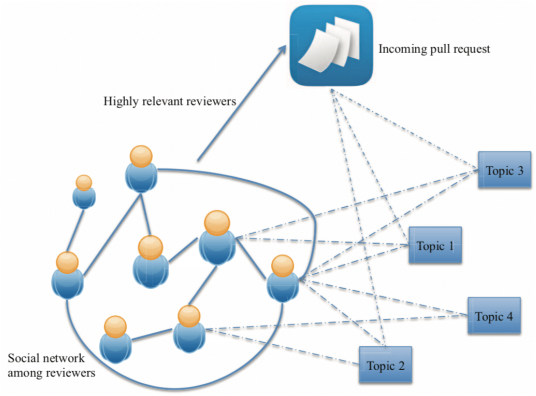

Figure 1: Graph of PR and Reviewer
Source: H. Ying et al. [7]

As in the Supervised Random walk (Leskovec et al) [1], We are given a snapshot of a network between pull requests and reviewers and would like to infer which pull requests will be handled by which reviewer among existing reviewers. Although the link prediction[6] problem has been extensively studied, the challenge of effectively combining the information from the network structure with rich edge attribute data remains mostly open. Our focus will be on analysis of selecting or creating edge features to help predict the correct reviewer for the given pull request.

In our results section, we will be analyzing the performance of existing Supervised Random Walk implementations and compare with our Graph Neural Network implementation. We will reason about its success/failure.

Problem Formulation : We are given a bipartite graph G in which, on one side, there are pull requests, and on the other side, there are reviewers. Pull requests are represented by set P, and reviewers are represented by set R. For a specific pull request, reviewers R are further divided into two categories, D - destination reviewers and L - no link reviewers, R = D ∪ L. For each edge (p, r), we create a feature vector called ψ_{pr} that describes the relationship between

node p and r.

For each edge (p, r), we compute edge strength $a_{uv} = f_w(\psi_{uv})$ where f_w is parameterized by w. For node u, v R, we have $p_u > p_v$ defined as the edge strength between source node s is stronger to candidate Reviewer u than candidate Reviewer v. In supervised Random Walks, these are Personalized PageRank[2] scores for s. The personalized PageRank scores satisfy the following condition: $p^T = p^T Q(w)$ where Q(w) is defined as:

$$Q(w) = \begin{cases} \frac{a_{uv}}{\sum a_{uv}} & \text{if } (u, v) \in G \\ 0 & \text{otherwise} \end{cases} \quad (1)$$

We want to optimize the parameter w, so that the model generates edge weights which give more bias to destination nodes than no link nodes. Hence the optimization can be viewed in terms of a distance maximization problem where we want to increase the distance between destination nodes and no-link nodes. The optimization problem will look similar to the constrained SVM problem :

$$F(w) = \|w\|^2 + \lambda \sum_{d \in D,\ l \in L} h(p_l - p_d) \quad (2)$$

The above objective function can be solved using iterative gradient descent. More implementation details are given in the literature [1].

We used the pull request data between 2018-01-01 and 2018-06-01 to build a graph G.We select a subset of pull requests P from G and use it as our training set. Each pull request in P has a set D of reviewers that it will be linked-to in the future. Note that we are using a complete set of reviewers in our training set.

The dataset we have prepared has around 11473 pull requests and 742 reviewers, so the graph we create is sparse because a particular pull request has been handled by a minimum of 1 reviewer and a maximum of 9 reviewers. We can clearly see that out of 806 reviewers, we have only 1-9 possible destination nodes and around 733-741 no-link nodes. This is a case of imbalance class distribution and should be handled with care.

To solve the above-mentioned objective function F(w) with high class imbalance in data, we need a good set of features to maximize the distance between the destination nodes and the no-link nodes. By maximizing distance we mean, to give more bias/preference to the selection of edges which lead to destination nodes than no-link nodes.

According to a recent implementation of SRW, 5 edge features are selected are used to train Supervised Random Walk.

1. Number of comments made by Reviewer r on Pull Request p: This feature makes sense because it signifies the amount of interaction between a pull request and reviewer. It has the highest weightage in recommendation of reviewer for a pull request p.

2. Number of total Pull-Requests submitted by Reviewer r to the repository: This feature signifies the authority and expertise of reviewer, if a reviewer submits a pull request to repository then there is a high possibility of reviewer submitting similar kind of pull request.

3. Number of different Pull-Requests commented of by Reviewer r: This feature represents the flexibility of reviewer in different areas of pull request, but this feature can sometime degrade the performance of SRW. It may happen that one reviewer is handling various type of pull requests and another reviewer who is handling only one type of pull request then our model can sometime recommend the first reviewer who has more flexibility instead of dedicated reviewer for that type of pull request. It can eventually lead to load imbalance in reviewer recommendation.

4. Number of days between the time the Pull-Request was submitted and the time the first comment was made: It represent about the preference of pull request i.e if a pull request is not handled by any reviewer for a longer duration

then that pull request is given less importance than the new pull requests submitted or vice-versa.

5. Number of days between the time the Pull-Request was submitted and the time the last comment was made: It represent the inactiveness of commentors on pull request. The inactive pull request is given less importance as compared to active pull request.

We did not implemented SRW in this entirety, instead we created a prototype of SRW algorithm to analyse the features and the variables affect on reviewer recommendation.The SRW prototype can easily be converted to complete SRW by just changing the dimensions of few variables.

4.2 Graph Neural Network

Problem Formulation : The basic intuition of our project builds upon present in Expertise-Authority Recommendation (EARec) Network defined by Ying et al.[7] Basically intuition behind graph based approaches proposed by many authors of building a comment network from available interaction between the reviewers and pull request owners and also intuition is developed from ideas behind non-graph based approaches such as File Location-based recommendation, Information Retrieval based approaches and we combined all knowledge together in this project. We explored the combination of IR + FL + CN. We inferred reviewer's expertise from their interactions with technically similar Pull-Requests. We build a graph that is consisting two type of nodes Reviewer and PR. Edges between them are representing relation between them. Comment Network based approach captures common interest from social interaction. Information retrieval and File location based approach captures expertise by taking into account the similarity factor. Here we are combining expertise and common interest together to make it more robust. Textual semantic information of PR extracts from its title and description. An incoming Pull-Request is added to network by calculating its similarity with other available previous PRs on the features extracted from title, description, file paths and it will get connected with top 20 most similar PR. To recommend top k reviewers we perform link prediction task between all reviewers and our PR and recommend reviewers with highest link probability using Graph Neural Network model GraphSAGE. To obtain node representation, GraphSAGE uses neural network to aggregate information from neighbors recursively by limited BFS. It gave embedding for each node and also incorporate node feature information. Graph topology contain useful information about the data. We used GraphSAGE because it is inductive framework and can be easily generalized on unseen data. So it is a dynamic approach which can generate embedding for unseen PR. That is very useful for our usecase because every time when a new PR came, we can not train our model again and again. Can think of it as in Neural network and similar way it produces output for a test data very quick. In the same manner it generates embedding for a new PR very quick.

GraphSAGE Algorithm :

We explored this graph using GraphSAGE GNN Model [5] to perform link prediction task between incoming PR and reviewer. we further pre-processed the dataset of examples and labels to construct an edge list, where an edge exists between PR and reviewer if and only if reviewer made at-least one comment on that PR. Using this networkx graph we constructed graph for the training examples as input and the corresponding ground-truth feature vectors as labels, we built a GraphSage GNN model consisted of message passing (i.e. aggregation) layers using mean aggregations. The model was trained with an ADAM optimizer and using the standard StellarGraph library, binary cross-entropy loss with logits as an objective function.

Graph Construction: We constructed graph using networkx python library, we have two types of nodes in our graph Reviewer and PR. Reviewer node is containing feature set from reviewer's github profile such as username, follower count, total github commits, base repo commit count. We found these features to be helpful to compute expertise of reviewer and PR

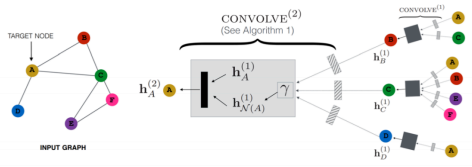

Figure 2: GraphSAGE

node contains features such as additions, changes, deletions and filepath count. We defined relationship between these nodes to compute edge weights between them.

Reviewer — Reviewer Relationship: Comments provided by developer on other developer's PR is connecting edge between them. If some reviewer r_i commented on PR of r_j reviewer and there will be an edge between them in the graph and edge weights will be calculated by number of comments or degree of participation by the method proposed by similar to Yu et al. [7] . Formally we iterate over all the PR submitted by user r_i and calculated comments on those PRs by user r_j and multiplied them by a decaying factor, basically we gave more importance to new comments then the older once. the authority among developers are extracted from comment relations between contributors and reviewers in previous PR.

PR — PR Relationship: Pull request relation with other pull request is defined by intuition behind Information retrieval based approach. First we calculated similarity between all pull requests to find out technically similar pull request and also connected one particular PR with top 20 similar PR. Textual sematic information of PR extracts from its title, description, file paths. We used language based character model chars2vec to convert these texts into numerical vectorize text-based features file paths and pull requests' titles and find similarity using cosine similarity using sklearn based implementation, previous paper approaches used techniques like TF-IDF or Latent Semantic Index models to do this task. Because chars2vec can capture errors without compromise. We assigned edge weights as similarity score between two PR. We can capture common interest developers by connecting PR this way.

PR — Reviewer Relationship: We connected reviewer to who made at-least one comment on particular PR and edge weights are computed based upon participation of reviewer on that PR. the weight assignment between incoming PR and a reviewer r_i can be computed as the comments counts of user r_i weighted by how semantically similar the incoming Pull-Request is to all Pull-Requests that r_i has reviewed in the past. A large edge weight signifies a strong expertise preference of reviewer.

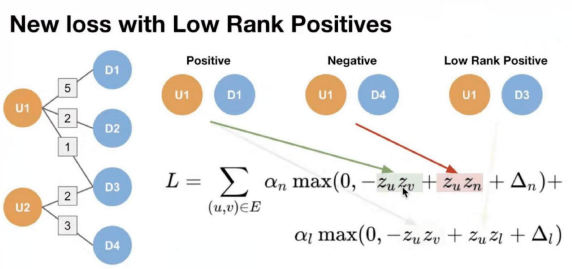

Figure 3: GraphSAGE Loss

5 Results:

5.1 Hyperparameters and SRW

λ is a hyperparameter used to do a tradeoff between low training error and overfitting.

$$F(w) = \|w\|^2 + \lambda \sum_{d\in D,\ l\in L} h(p_l - p_d) \qquad (3)$$

From the above equation, we can clearly see that $F(w)$ is directly proportional to λ , so we experi-

mented with different values of λ and found that on decreasing λ, the loss was decreasing.

When we used one example to train the model then taking small lambda was quite sensible because the loss function is directly proportional to lambda.While taking large set of training examples, taking a large value of lambda performs better to avoid overfitting. We also took one feature at a time to analyze its performance and to select best set of features which works best with λ.

There is also a hyperparameter α which is the coefficient for restart of walk in graph. Since we just need to predict one edge so we tried using model with or without alpha. For few training examples, the model without alpha works fine and produced desired result but with large set of training examples, taking α into consideration worked better.

5.2 Evaluation Metrics

We evaluate the performances of our approaches over each project by precision, recall and F-Measure which are widely used as standard metrics in previous work. In our case precision is basically percentage of reviewers that we suggested who actually commented on that particular PR and recall is percentage the real reviewers who are actually recommended. F-Measure considers both the precision and the recall of the test.

$$\text{Precision} = \frac{1}{m}\sum_{i=1}^{m}\frac{|\text{Actual Set}_i \cap \text{Rec Set}_i|}{|\text{Rec Set}_i|} \quad (4)$$

$$\text{Recall} = \frac{1}{m}\sum_{i=1}^{m}\frac{|\text{Actual Set}_i \cap \text{Rec Set}_i|}{|\text{Actual Set}_i|} \quad (5)$$

$$F1 = \frac{2 * \text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}} \quad (6)$$

We performed experiments to extract features from multiple sources and converted them into appropriate form by using language text models as *chars2vec* and then defined node features and edge weights and also performed experiments to if they are performing well as used approaches defined in research papers and mixed multiple approaches and performed to see which is working better with reasons also. Finally we made our link prediction model over multiple such approaches and got better reseults then all baseine Non-graph and Graph based approaches.

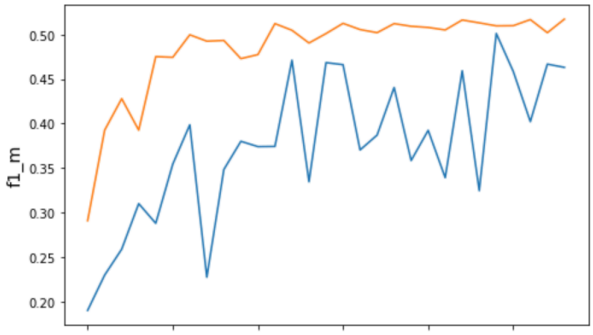

Figure 4: F1 Score GNN
Train Test

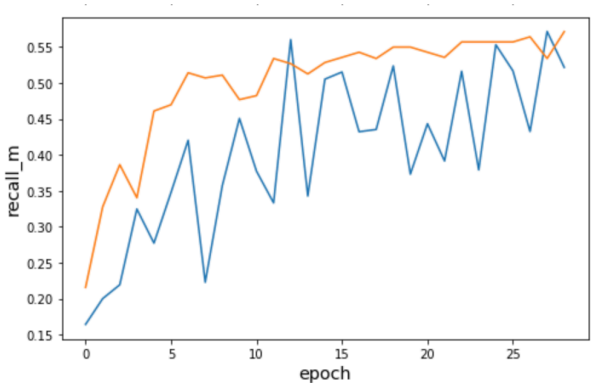

Figure 5: Recall Score GNN
Train Test

Method	F-1 score	Precision	Recall
Activeness	0.108	0.071	0.286
Expertise	0.149	0.096	0.405
Bipartite SRW	0.173	0.110	0.434
Graph NN	0.520	0.478	0.570

Table 4: Comparision of GNN with other models

6 Conclusion and Future work:

We propose a novel approach for reviewer recommendation to speed up the review process and reduce the stress of core developers. Large amount of incoming pull-request challenge developers and increase waiting time. we mined the publicly available data on GitHub for the Kubernetes repository from various resources such as Github API, GHTorrent [4] and GHArchive, created a data set to recommend Reviewers to Pull-Requests. Evaluate their performances on Kubernetes of GitHub using precision, recall and F-Measure. Analysed our results with other baseline approaches. Integrating reviewer's expertise and comment interests, represent significant improvements in precision and recall, and the overall performances of mixed approaches are more stable than using different approaches independently. The results indicate that combining the social factors (e.g., common interests among devel- opers) and technical factors (e.g., developers expertise).. we frame this problem as a link prediction task, and experiment with Graph Neural Network. We constructed graph with two nodes, reviewer and PR. Reviewers are connected with each other based upon weather they commented on each others PR and PRs are connected based upon text semantic similarity. Then we predicted link between Reviewer and PR compensating the expertise, authority and common interest together. In our future work, we plan to explore the combinations of Information Retrieval, File Location, Comment Network with some different feature set such as using graph attributes such as node degree, shortest path etc. as features in down stream supervised classifiers for reviewer node and PR node.

7 Member Contributions:

Name:	Harshvardhan Pratap Singh
Roll No:	20111410
Programme:	MS - CSE

- Literature Survey.
- Problem formulation and defined end goal.
- Defined architecture of network by experiments and analysis.
- Data Preprossing {Data Modeling, Feature Extraction, Feature Embedding, Graph Construction}
- Contructed Graph Neural Network Model and Analysis

Name:	Yash Saraswat
Roll No:	20111073
Programme:	MTech - CSE

- Literature Survey
- Feature Extraction, Feature Analysis for Supervised Random Walk
- Developed Supervised Random Walk prototype and Analysis

Name:	Gaurav Tank
Roll No:	20111407
Programme:	MS - CSE

- Literature Survey.
- Fetching the dataset from BigQuery.
- Scrapping the PR file changes and PR titles.
- Dataset formatting and information extraction from the dataset.

8 Github Repository Link:

Our codebase is available at github.code

References

[1] Lars Backstrom and Jure Leskovec. "Supervised Random Walks: Predicting and Recommending Links in Social Networks". In: *CoRR* abs/1011.4071 (2010). arXiv: 1011.4071. URL: http://arxiv.org/abs/1011.4071.

[2] Bahman Bahmani, Abdur Chowdhury, and Ashish Goel. "Fast Incremental and Personalized PageRank over Distributed Main Memory Databases". In: *CoRR* abs/1006.2880 (2010). arXiv: 1006.2880. URL: http://arxiv.org/abs/1006.2880.

[3] Mukund Deshpande and George Karypis. "Item-based top-N recommendation algorithms". In: *ACM Trans. Inf. Syst.* 22.1 (), pp. 143–177. ISSN: 1046-8188. DOI: 10.1145/963770.963776. URL: https://doi.org/10.1145/963770.963776.

[4] G. Gousios. "The GHTorent dataset and tool suite". In: *2013 10th Working Conference on Mining Software Repositories (MSR)*. 2013, pp. 233–236. DOI: 10.1109/MSR.2013.6624034.

[5] Will Hamilton, Zhitao Ying, and Jure Leskovec. "Inductive Representation Learning on Large Graphs". In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon et al. Vol. 30. Curran Associates, Inc., pp. 1024–1034. URL: https://proceedings.neurips.cc/paper/2017/file/5dd9db5e033da9c6fb5ba83c7a7ebea9-Paper.pdf.

[6] T. Lee and Wen-Chien Chen. "Applying Link Prediction for Repository Recommendation on GitHub". In: 2015.

[7] H. Ying et al. "EARec: Leveraging Expertise and Authority for Pull-Request Reviewer Recommendation in GitHub". In: *2016 IEEE/ACM 3rd International Workshop on CrowdSourcing in Software Engineering (CSI-SE)*. 2016, pp. 29–35. DOI: 10.1109/CSI-SE.2016.013.

[8] Yue Yu et al. "Reviewer recommendation for pull-requests in GitHub: What can we learn from code review and bug assignment?" In: *Information and Software Technology* 74 (2016), pp. 204–218. ISSN: 0950-5849. DOI: https://doi.org/10.1016/j.infsof.2016.01.004. URL: http://www.sciencedirect.com/science/article/pii/S0950584916000069.