Github Reviewer Recommendation using Graph

Final Report - CS224W Machine Learning with Graphs

Toan Luong
Stanford University
toanlm@stanford.edu

Ryan Silva
Stanford University
rdsilva@stanford.edu

Apollo Kaneko
Stanford University
akkaneko@stanford.edu

***Abstract*—Pull-Requests play a central role in Github as a premier social coding platform and facilitate the technical communication between project members and community contributors. For large-scale and popular projects, the number of Pull-Requests are substantially high and can overwhelm the core team maintenance schedule, which can lead to late responses and appropriate Pull-Requests being ignored. For the project, we construct the Expertise-Authority Recommendation (EARec) Network with additional modifications on reviewer-reviewer edge weights and Pull-Requests' content similarity measure. We perform recommendations using this network via Random Walk with Restart [2] and compare the performance with baselines from the literature including Activeness and Expertise models. We then explore Supervised Random Walks to perform link prediction on a bipartite graph between Pull-Requests and Reviewers as a second method of recommendation. We find that both methods show reasonable improvement though training and hyper parameter tuning, but are limited due to the amount of computation needed.**

***Index Terms*—github, pull request, reviewer, recommendation, supervised random walk, graph**

I. Introduction

As the largest open-source community hosting thousands of software repositories, Github popularizes the pull-based development model [6] where external contributors outside the core team can propose source code changes and resolve issues. A contributor (individual or group) works on their independent *fork* of the *base* repository until the code changes are mature and ready to be merged to the official base development branch. To formalize this process, a *Pull-Request* is submitted, which details the issue(s) the code changes aim to resolve and how they are implemented. With built-in capabilities that highlight code changes, Pull-Requests in Github facilitates the technical communications between project contributors in the form of code reviews, style feedback, and general problem discussions. For a large-scale project , the volume of social interactions over thousands of Pull-Requests that span several years create an interesting network with tight-knit communities to explore.

According to Gousios et al. [6], for popular projects, the number of Pull-Requests are substantially high and can overwhelm the core team maintenance schedule, which can lead 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. Oftentimes, a Pull-Request submitter tends to tag other contributors whom they worked with in the past or popular core team members as *Reviewers*. Although a core project member with write privileges is always needed, a technical discussion in a Pull-Request can benefit from any experienced developers outside of the core members social circle. The process can benefit from a Reviewer Recommendation System where reviewers can be reminded of Pull Requests relevant to their expertise and priorities. If effective, such a system can save reviewers' time and improve overall code quality due to short feedback loop time.

Drawing inspirations from the literature on reviewer recommendation problem [17], [16], [15], [7], and graph-based solutions for content discovery and personalized recommendations [4], [1], [14], the project aims to validate the literature findings on a large-scale recent project such as **Kubernetes** (*kubernetes/kubernetes* on GitHub). We apply two approaches to Reviewer recommendation: first, we update several elements in the Expertise-Authority Recommendation Network [15] and evaluate the performance of Personalized Random Walk recommendations on this network. Second, we apply Supervised Random Walks (SRW) [1] to a bipartite graph between Reviewers and Pull-Requests to learn an edge strength function which maps edge features to edge weights. We then run Personalized Random Walk in this weighted bipartite graph to produce recommendations. In the EARec model, we are able to recommend Reviewers on a completely unseen Pull-Request, whereas SRW requires that the new Pull-Request has had Reviewers submit comments, and more Reviewers are recommended based on the previous Reviewers.

II. Related Work

Pull-Request Reviewer Recommendation Problem refers to an automated system that recommends reviewers, which can include core project team members, veteran contributors, or domain experts, to review a Pull-Request in depth. Although the submitter can use the *@mention* function to alert relevant reviewers, the literature acknowledges evaluating Pull-Requests in an efficient and timely manner is still a complex and challenging problem. Several publications study the nature of Pull-Requests and reveal several socio-technical attributes of the submitters and the code content to critically influence the evaluation process and acceptance latency. Gousios et al. [6] show that the time latency between Pull-Request creation

and acceptance is dependent on the developers' track record, project size, test coverage, and project contributing policy. Tsay, Dabbish, and Herbsleb [13] argue that both technical and social signals have strong associations with Pull-Request acceptance. Interestingly, Pull-Requests with many comments were much less likely to be accepted, yet the negative influence could be mitigated by the submitter's prior interaction in the project and track record. These findings validate our approach of considering both developers' expertise and their relationships with other developers in the recommendation engine.

Content-Based Pull-Request Reviewer Recommendation.. A significant body of literature focuses on information retrieval techniques and traditional recommendation methods to associate reviewers with the source code content in the Pull-Requests. Since a Pull-Request contains rich features including the number of lines that have been added and removed, files' names, locations, and commit history trees, these methods yield relative successes. Thongtanunam et al. [12] file-location approach where similar file paths should be reviewed by similar reviewers recommended 79% of reviews with top 10 candidates and a median rank of 4. Rahman, Roy, and Collins [11] also incorporate data about external libraries and specific technologies related to the source code changes to infer similarities between new and historical Pull-Requests. Jiang et al. [7] adopt a SVM classifier on predictors related to social relationships, file locations, and recent activities of core team members, and achieve accuracy ranging from 49.3% to72.3% for Top-1 recommendation, and from 72.9% to 93.5% for Top-3 recommendation. Similar performance are also yield when Lima Júnior et al. [9] use a Random Forest model. These findings validate our approach of considering Pull-Request content similarity and establish the different recommendation benchmarks that our approach can be compared to.

Graph-Based Pull-Request Reviewer Recommendation. Our project draws significant insights from Yu et al. [16] and Ying et al. [15]. The interactions between core members, external commenters, and code contributors in a particular project form a dynamic social network, on top of the rich information from their Github profile pages and involvement with other projects. Utilizing the network structure 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, as opposed to the limited set of core developers. Yu et al. [16] build the Comment Network as a directed, weighted graph between GitHub users. Edges are created between users if a user i has reviewed at least one previous PR by user j. The weight of these edges is determined by the number, distribution, and time of the comments. Ying et al. [15] utilizes a Latent Semantic Index (LSI) model to assign edge weights between a Pull-Request and and a user based on their interactions on other textually similar Pull-Requests. Graph-based approaches yield similar F-measures compared with information retrieval and file level approaches. The authors recognize that an ensemble approach can be more stable than using different approaches independently. Construct, internal, and external validity related to data sample bias, non-considerations of other network, and generalization to other social coding platforms are also discussed. Notably, recommendation performances can be varied across different projects in Github and hint at the instability of graph-based approaches for recommendation problems.

Our method is an extension of the Comment Network proposed by Yu et al. [16] and Expertise-Authority Recommendation Network proposed by Ying et al. [15]. For a given large-scale repository, we construct a directed, weighted network between reviewers based on their code review and comment interactions. An incoming Pull-Request is added to the network by approximating the similarity of the Pull-Request to previous ones. Hand-crafted features related to code modifications, file paths, title, and description are utilized to assess the content similarity between an incoming Pull-Requests and other Pull-Requests commented by User nodes in the graph. To recommend top-k reviewers for an incoming Pull-Request, we adopt the Supervised Random Walk [1] and benchmark with a popularity-based baseline approach and the Random Walk with Restart method proposed by the authors in [15]. Another contribution to the literature is that our experiments and validations involve a newer data extract from GHTorrent [5], GHArchive, Github API with recent massive-scale projects with modern engineering standards such as Kubernetes.

III. Approach

In this section, we describe the Expertise-Authority Recommendation (EARec) Network [15] with additional modifications related to similarity measures and edge weights between Reviewers. With the defined graph, we detail the following approach to recommend Reviewers: popularity baseline, Random Walk with Restart, and Supervised Random Walks. The below discussion context is limited to a *single* target repository due to the weak connectivity between users across different repositories.

A. Activeness Random Selection

Most open-source projects are managed by a team who frequently commit, submit issues, or participate in pull requests. An obvious baseline recommendation heuristic for an incoming pull request can be recommending the top-k most active developers. Their "activeness" rankings are determined by the number of pull requests the had submitted or commented before the new pull request's creation date. This is a similar baseline to Yu et al. [16].

B. Collaborative Filtering using Expertise Scores

As a non-graph benchmark, we employ an item-item collaborative filtering technique propose by Deshpande and Karypis [3] to recommend reviewers for an incoming pull request. Let P_{test} and P_{train} be pull requests, we define a similarity metric $S(P_{\text{test}}, P_{\text{train}})$ to reflect their content similarity. In literature, features for content similarity include pull requests' titles, descriptions, file paths, which are vectorized by several methods

such as TF-IDF or Latent Semantic Index models. In our project, we mine similar information about the pull requests from the Github API for the Kubernetes project and employ a character-based language model *chars2vec* implemented by Intuition Engineering Team to vectorize text-based features file paths and pull requests' titles. Since they are filled with code blocks, quoted replies, and other technical jargon, *chars2vec* can capture the content similarity without compromising for out-of-vocabulary errors of other word embedding models. Since *chars2vec* implementation uses Tensorflow and GPU processing, similarity calculations between Numpy vectors can be much more efficient than naive string comparisons proposed by Thongtanunam et al. [12]. After the feature vector is crafted and standard scaled for each pull request, we compute a pairwise similarity matrix using Cosine Similarity using Scikit-Learn efficient implementation.

Next, let $N(P_{\text{train}}; k)$ as the set of k most "similar" in content to the incoming pull request P_{test} as defined by the crafted feature vectors and similarity metric. For each $P_{\text{train}} \in N(P_{\text{train}}; k)$, we calculate the cumulative number of occurrences of of each reviewer and use that score to determine their recommendation ranking.

C. Expertise-Authority Recommendation Network (EARec)

Our project reconstructs most elements of the Expertise-Authority Recommendation (EARec) Network defined by Ying et al. [15] with additional modifications on Reviewers' edges weights and textual similarity measure. When matching a Pull-Request with a potential Reviewer, EARec takes into account 1) the Reviewer's *expertise* inferred from their interactions with technically similar Pull-Requests, and 2) the Reviewer's *authority* in the social network of all contributors in the repository. EARec graph consists of two node types: $N+1$ nodes with N candidate Reviewers and one incoming Pull-Request S as the propagation source node. Within a fixed timeframe and a base repository, we consider the set of all Github users who comment in at least one of the repository's Pull-Requests to be the candidate Reviewer set $R = \{r_1, r_2, ..., r_N\}$. Similar to the authors' definition, the graph can be denoted as $G = (V, E, W)$. The node set $V = R \cup S$, the edge set between Reviewers is $E = \{e_{ij} | 0 < i, j \leq N+1\}$, and the edge weight set $W = \{w_i j\}$ denotes the social relationship or *authority extent* between Reviewer i and other Reviewers in the candidate set.

Reviewer − Reviewer Relationship. Different from the original EARec, we define the Reviewer network as a weighted directed graph. Similar to Yu et al. [16], there is an edge between a Reviewer node r_i and another Reviewer r_j if r_j participates in at least one of r_i's Pull-Requests. The degree of participation is reflected in the edge weight w_{ij} and takes into account the age, frequency, spread across all Pull-Requests that user r_i has submitted. Formally, given the set PR_i of Pull-Requests submitted by user r_i, by iterating through each Pull-Request s, we accumulate the number of comments made by user r_j but multiplied by a decaying hyperparameter of λ and time-sensitive factor $t_{(ij,s,c)}$. Intuitively, we value r_j's comments in multiple Pull-Requests of PR_i rather than "spamming" on a handful. New comments from r_j are more valuable than their old ones controlled by $t_{(ij,s,c)} \in (0, 1]$.

$$w_{ij} = \sum_{s=1}^{|PR_i|} w(ij, s) = \sum_{s=1}^{|PR_i|} \sum_{c=1}^{|\text{comments}|} \lambda^{c-1} \cdot t_{(ij,s,c)} \tag{1}$$

Pull-Request − Reviewer Relationship. Similar to Ying et al. [15], let the set of all Pull-Requests *reviewed* by a candidate reviewer r_i as $S_i = \{s_i^1, s_i^2, ..., s_i^j, ...s_i^{|S_i|}\}$. For an incoming Pull-Request s_{new}, we compute the edge weight between s_{new} and a candidate reviewer r_i by considering r_i's participation in all Pull-Requests that has similar content to s_{new} represented by $S_i' \subset S_i$. While the authors use Latent Semantic Index (LSI) model, we re-use the feature vectors crafted for Collaborative Filtering baseline described in III-B which utilizes a character-embedding model *chars2vec* along with other metadata related to number of additions, deletions, and changes for each edited file in the pull request. To approximate the content similarity between Pull-Request s_{new} and $s_i^j \in S_i$, we utilize cosine similarity that is formally expressed in the below equation:

$$\text{sim}(s_{\text{new}}, s_i^j) = \frac{\text{vec}_{s_i^j} \cdot \text{vec}_{s_{\text{new}}}}{\text{vec}_{s_i^j} \times \text{vec}_{s_{\text{new}}}} \tag{2}$$

Hence, the weight assignment between s_{new} and a candidate reviewer r_i can be computed as the comments counts of user r_i weighted by how semantically similar the incoming Pull-Request s_{new} is to all Pull-Requests that r_i has reviewed in the past. A large edge weight signifies a strong expertise preference.

$$w_{(s_{\text{new}}, r_i)} = \frac{\sum_{j=1}^{|S_i|} n_{s_i^j} \cdot \text{sim}(s_{\text{new}}, s_i^j)}{\sum_{j=1}^{|S_i|} n_{s_i^j}} \tag{3}$$

D. Reviewers Recommendation Method

Below, we present the propagation approach over the constructed bipartite graph to recommend reviewers for an incoming Pull-Request. Intuitively, the baseline recommendation method is to recommend the top-k active reviewers in the repository, judged by the number of comments and Pull-Requests participated, up to the Pull-Request creation date.

Random Walk with Restart (RWR). Following Ying et al. [15], we use personalized RWR, also known as personalized PageRank, to make recommendations from the EARec network. Briefly, the RWR algorithm finds the stationary point $\vec{u}_{s_{\text{new}}}$ of the equation

$$\vec{u}_{s_{\text{new}}} = (1 - \alpha) A \vec{u}_{s_{\text{new}}} + \alpha \vec{v}_{s_{\text{new}}} \tag{4}$$

A is the stochastic adjacency matrix of the EARec Network, $\vec{v}_{s_{\text{new}}}$ is the personalization vector with a non-zero entry at node s_{new} and zeros otherwise, and α is the probability of restarting the random walk from the source node s_{new}. The stationary point is computed by iteratively applying the transition matrix to $\vec{u}_{s_{\text{new}}}$ until convergence. At this point, we can rank and

recommend Reviewers form the network using their score in $\vec{u}_{s_{\text{new}}}$.

E. Supervised Random Walks

Proposed by Backstrom and Leskovec [1] and also described in detail in [8], the Supervised Random Walk algorithm is a hybrid approach that combines both node features and edge features to learn *edge strengths* to perform a link prediction task. The objective of Supervised Random Walk is to learn a mapping from edge features to edge weights, such that a ranking of nodes by personalized random walk will rank nodes in the label set higher than those in the rest of the graph.

Given a graph $G = (V, E)$, for a source Pull-Request node $s \in V$, we can define the set of candidate nodes for future links of s as the candidate Reviewers set $R = \{r \in R | (r, s) \notin E\}$. We also define a future destination set of candidate Reviewers $D \subset R$, which is the set of links that the algorithm aims to predict, and the no-link set $L = R - D$. For node $u, v \in R$, we have $p_u > p_v$ defined as the affinity strength between source node s is stronger to candidate Reviewer u than candidate Reviewer v. In supervised Random Walks, these are Personalized PageRank scores for s. The graph which these scores come from is defined by G, the features fore each edge in G $z_{u,v}$ and an edge strength function:

$$a_{uv} = f_w(z_{u,v}) \tag{5}$$

Therefore the Personalized Pagerank scores satisfy the stationary condition

$$p^T = p^T Q(w) \tag{6}$$

where $Q(w)$ is defined as

$$Q_{uv}(w) = \begin{cases} (1-\alpha)\frac{a_{uv}}{\sum_k a_{uv}} + \alpha \mathbf{1}(v = s) & (u, v) \in E \\ \alpha \mathbf{1}(v = s) & otherwise \end{cases}$$

We want to learn a set of parameters w that are optimized to produce edge weights which bias Personalized PageRank to rank nodes in D higher than L. Hence the optimization problem can be set up to minimize the occurrences when $p_d < p_l$ which is undesirable. In 1, h is the penalty function and L2-norm regularization is added to avoid overfitting:

$$\min_w F(w) = \sum_{d \in D, l \in L} h(p_l - p_d) + \lambda ||w||^2 \tag{1}$$

This objective function is optimized by iterative gradient descent until convergence. See Backstrom and Leskovec [1] for implementation details for calculating the gradient with respect to w. In our implementation we calculate these gradients and use an L-BFGS solver to optimize the weights.

Supervised Random Walks Experimental Setup

For our approach, we propose a new graph structure that is a bipartite graph between Reviewer nodes and Pull-Request nodes, in order to take advantage of the rich features available between Reviewers and Pull-Requests in our data set. In this graph, links between nodes correspond to a Reviewer commenting on a Pull-Request and are unweighted.

We produce a set of training examples from a snapshot of the graph between 2018-01-01 and 2018-06-01, G, and a snapshot of the graph between 2018-01-01 to present, G_{future}. We select a set of source Pull-Requests S from G which serve as supervised training examples. Each Pull-Requests in S has a set D of Reviewers that it will be linked to in G_{future}. Some Reiewers in the dataset never make a comment until after 2018-06-01, which means the respective Reviewer nodes would be unconnected in G, and thus we omit these Reviewer nodes from our experiment.

In our setup we require that each node in S has a minimum of 2 links in G and 2 nodes in D. There are 133 source nodes which meet this requirement in our experiments. The average number of existing links in G for these source nodes is 3.6, with the max being 11, and the average number of nodes in D is 2.9, and a max of 9. Each source node also has a no-link set L, which is comprised of negative samples in G_{future}. In our experiments, L is the set of Reviewer nodes in G and not already in D.

We compute the following edge features from the data for each edge (r, p):

- Number of comments made by Reviewer r on Pull-Request p
- Number of total Pull-Requests submitted by Reviewer r to the repository
- Number of different Pull-Requests commented of by Reviewer r
- Number of days between the time the Pull-Request was submitted and the time the first comment was made.
- Number of days between the time the Pull-Request was submitted and the time the last comment was made
- The cosine similarity feature used in the EARec network, explained in section .

We use the logistic function for the edge strength function and the Wilcoxon-Mann-Whitney (WMW) loss function:

$$h(x) = \frac{1}{1 + exp(-x/b)}$$

At test time, using the trained edge strength function, we generate edge weights on a graph with an incoming Pull-Request s_{new}. We then run Personalized PageRank [10] to produce a ranking of nodes in the graph for s_{new}. Filtering out the Pull-Request nodes and already connected Reviewer nodes yields rankings for set of candidate Reviewers who are likely to review the new Pull-Request.

IV. Experiments

A. Data

Due to the extensive effort to extract, transform, and load data from a variety of sources including GHArchive and Github API, we performed our experiments exclusively on the **Kubernetes** project. Kubernetes fits our analysis due to its

scale with more than 80,000 commits, 2000 core contributors, and 50,000 pull requests since its creation. All code reviews and technical discussions for the Kubernetes project occur on the Github platform unlike Apache projects that utilize Jira and other software management systems heavily. In practice, this is a major challenge of recommending Reviewers, since many interactions may not be documented or inaccessible.

We originally tried querying from GHTorrent project [5] available on Google BiqQuery, however only code review comments are available. Instead, we managed to combine several JSON tables on GHArchive to extract all code review and general issue comments along with metadata capturing the interactions between Kubernetes contributors. From the Github API, we extracted the list of file paths, number of additions, deletions, and changes involving each pull request. Our ETL pipelines are documented for future literature.

Due to scability issues of graph-based algorithms, we limit our analysis to pull requests and comments within 2018. Specifically, pull requests and comments made before June 2018 are reserved for training. Only commenters in the train set are considered for recommendations of test pull requests made after June 2018. As the result, the train set contains 31,064 comments of 2719 pull requests and 690 commenters who are all in the reviewers candidate set. We reserve 3837 pull requests for testing and only 366 commenters out of the candidate set present for recommendations. Fixing the candidate set to only "seen" commenters helps scale the problem appropriately for graph-based recommendations, since our proposed approaches do not accomodate evolving graph structures.

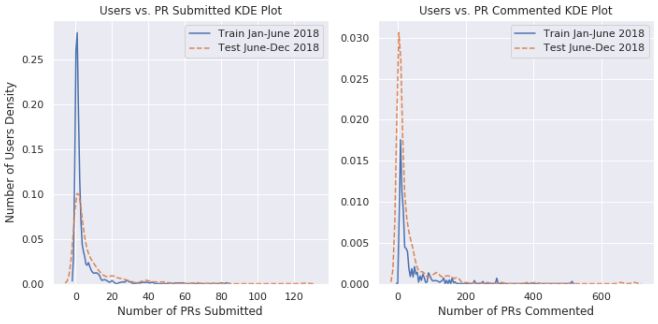

Fig. 1. PR-Users Distribution Plot

Figure 1 and 2 demonstrate the similarity between train and test set distributions in several interaction aspects between pull requests and participants of the Kubernetes project. Most users submit less than 20 pull requests for review, while the number of pull requests they participate in the technical discussion has a much wider range. In the both train and test sets, we see a handful number of users, who are core Kubernetes contributors, making more than 200 comments over the 6 months period. Most pull requests have less than 25 comments with an average between 3 and 4 commenters excluding the owners.

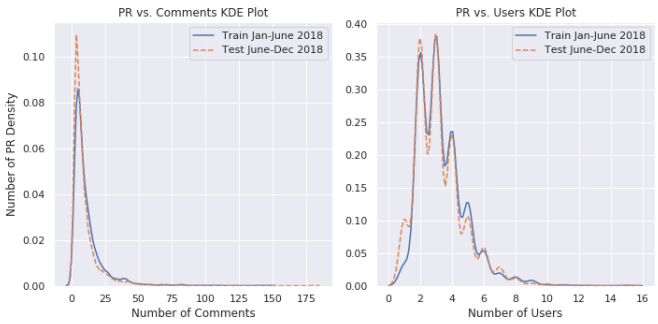

Fig. 2. PR-Comments Distribution Plot

B. Evaluation Metrics

For evaluation metrics, similar to the literature, we use top-10 precision, recall, and F-Measure to evaluate the reviewer recommendation system. As Jiang et al. (2015) [7] explained, "precision is the percentage of suggested top-k developers who actually comment on the pull request and recall is the percentage of real commenters who are actually suggested". We limit our guess range to 10 to accommodate the recommendation problem difficulty, while a higher guess range would be unrealistic. The authors suggest the following formula where m is the number of of test Pull-Requests.

$$\text{Precision@10} = \frac{1}{m}\sum_{i=1}^{m}\frac{|ActualSet_i \cap RecSet_i|}{|RecSet_i|}$$

$$\text{Recall@10} = \frac{1}{m}\sum_{i=1}^{m}\frac{|ActualSet_i \cap RecSet_i|}{|ActualSet|}$$

$$\text{F1} = \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

For a test Pull-Request i, the code contributor is not included in the $ActualSet_i$. If a developer leaves one or more comments, they still count as one reviewer in the $ActualSet_i$.

The $ActualSet_i$ is slightly different for EARec and SRW. EARec is able to make recommendations based solely on the content of the Pull-Request, whereas in SRW, we can only recommend new Reviewers after links in the graph have been formed. This means the $ActualSet_i$ for EARec includes all future Reviewers, while for SRW, in our experiments half of the actual Reviewers are in $ActualSet_i$, while the algorithm has access to the other half within the graph structure.

In addition, we also employ Mean Reciprocal Rank (MRR) to evaluate the correct reviewers' ranks in the recommendation sets, defined as:

$$\text{MRR} = \frac{1}{|P|}\sum_{p \in P}\frac{1}{\text{rank}_p}$$

V. Results

Below, we discuss the results of our experiments.

TABLE I
BEST RECOMMENDATION RESULTS

	F-1@10	MRR	Precision@10	Recall@10
Activeness	.108	.137	.071	.286
Expertise	.149	.194	.096	.405
EARec RWR	.094	.127	.062	.249
Bipartie SRW	**.173**	**.112**	**.110**	**.434**

A. Non-Graph Methods

As summarized in table V-A, Activeness Random Selection produces a low but hard-to-beat baseline of 10.8% F1 for top 10 predictions. This reflects the intuition that pull requests reviewers are highly technical and frequently visited by the same set of ”active” developers. The Collaborative Filtering approach indeed beat the baseline model with 14.9% F1. It reflects that pull request reviewers tend to stick with pull requests with similar content to what they already reviewed before.

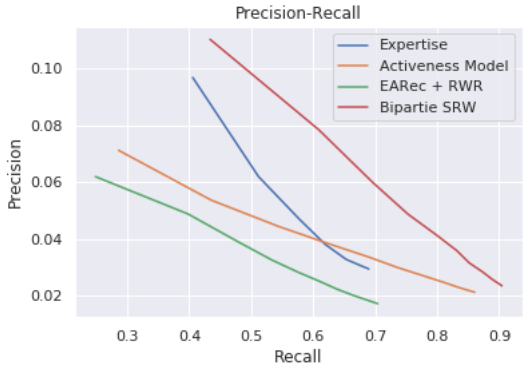

Fig. 3. Precision-Recall Plot between 10 to 100 prediction range

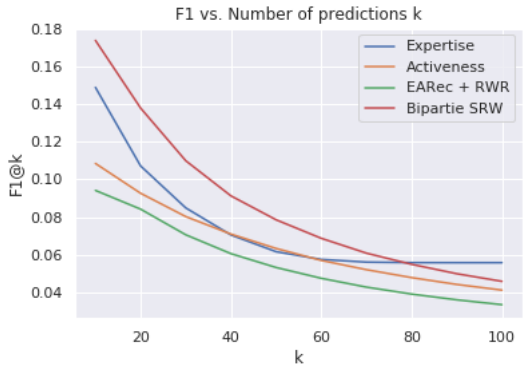

Fig. 4. F1 Plot for top k between 10 to 100 prediction range

B. EARec

After creating the EARec graph of Kubernetes reviewers, we found that the graph is weakly connected. We also found that 27% of the users in the graph only submitted a Pull-Request to the project and did not comment, while 3% of users only commented and did not submit a Pull-Request. Table II and graphs below summarize the statistics for the network.

TABLE II
KUBERNETES EAREC STATISTICS

Nodes	713
Edges	3714
Average Path Length	1.1069
Average Clustering Coefficient	0.1934

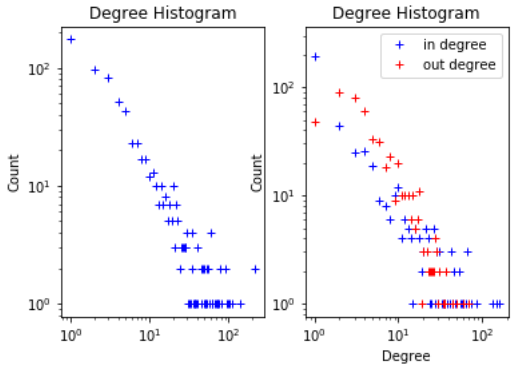

Fig. 5. Degree Histogram for EARec network

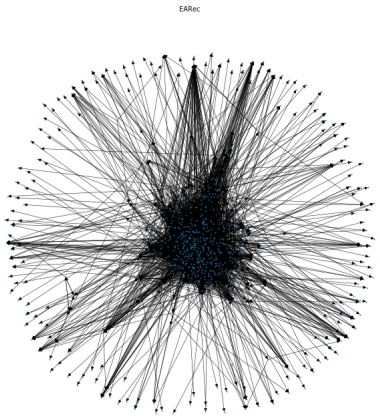

Fig. 6. A visual example of EARec network. Many contributors are on the edge of the network and are weakly connected, while there is a central core of developers in the center of the network

For each test Pull-Request, we perform Personalized PageRank on the EARec network we constructed for the Kubernetes project. We summarize the average statistics for the hyper-parameter search we performed over alpha in V-B. An alpha of 0 corresponds to ranking Reviewers solely based on their cosine-similarity metric used for the Pull-Request to Reviewer edge weight, while an alpha of 1 corresponds to no personalization and performing RWR on the Comment Network, which

TABLE III
RWR HYPERPARAMTER SEARCH ON THE EAREC NETWORK

Alpha	F-1	MRR	Precision@10
0	.004	.011	.003
0.25	.086	.119	.056
0.5	.094	.127	.062
0.1	.091	.126	.060
0.25	.091	.124	.060
0.5	.085	.124	.056
0.75	.084	.122	.056
0.9	.085	.119	.056
1.0	.083	.122	.055

produces the same set of recommendations for every Pull-Request. We find that an alpha parameter of .05 is optimal on our test set for both MRR and F-1 metrics.

In a sense, the alpha parameter controls how much the algorithm overfits to the content of the pull request versus how much it uses the developer interaction on a project wide basis. It is interesting to note that using neither the Reviewer-Reviewer or cosine-similarity scores by themselves work very well, however a heavily weighted personalization vector with minimal reliance on the network structure is optimal.

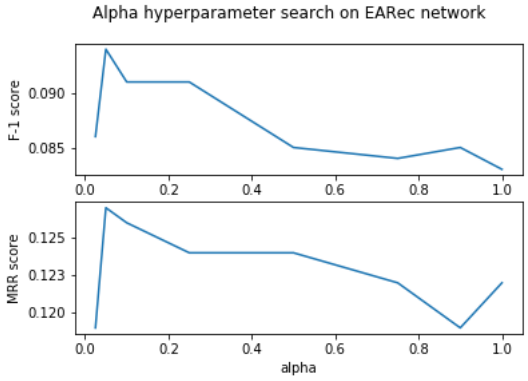

Fig. 7. Hyperparameter Search Performance

C. Supervised Random Walk

We set up our graph and necessary functions as described in our Approach section. As our data is quite large, we utilize sparse matrices in our implementation to calculate our gradients. We then use the L-BFGS algorithm to perform gradient descent. We set regularization value $\lambda = 1$ and restart probability $\alpha = 0.3$, in line with recommendations made in Backstorm 2010. Due to time constraints, we were not able to perform a hyperparameter search to fit our data. Nevertheless, we found that Supervised Random Walk performed the best out of the models we built, resulting in a better F-1 score for most k.

VI. DISCUSSION

From our results, we can see that our bipartite supervised random walk approach outperforms both the baselines and EARec model, while the EARec approach under performs. From this, we can assume that the supervised random walk algorithm learned edge strengths that were more optimal than the features computed for our EARec model. However, none of our approaches resulted in a F1-score above 0.2 using top-10 predictions. We discuss the limitation of our approach and dataset below.

First, as we do not attempt to predict how many users contribute a certain pull request and instead take the *top-k* predictions, the performance of our recall and precision depends on how close *k* is to the actual number of contributors. In future works, a separate model predicting the number of contributors for a pull request could improve performance.

Second, the *kubernetes* repository and Github pull requests in general have a relatively few number of contributors per pull request. Furthermore, we can see from our baseline that predicting the ”most active” or ”core” developers in the repository is not sufficient to get good prediction accuracy. As each user only contributes to a small subset of pull requests and each pull request has a small number of contributors, this recommendation task is difficult.

Finally, pull request comments most likely does not capture the entirety of the interactions between users. Discussions through other channels such as email or in person conversations can make a significant difference. This could be one reason that the broader more generalized baselines could outperform some graph methods. These three difficulties are similarly reflected in the performance of other research done on this task.

Our research is also limited in its breadth. As Github repositories vary in terms of management and popularity, the graph structure of each repository have significant differences. Our experiments only considered one repository, and therefore cannot signify how these models will generalize to other projects.

VII. CONCLUSION & FUTURE WORK

For our project, we mined the publicly available data on GitHub for the Kubernetes repository and create a data set for training algorithms to recommend Reviewers to Pull-Requests. Our task is to perform Reviewer recommendations for Pull-Requests within this repository and we frame this problem as a link prediction task, and experiment with two different methods.

First we propose an update to the EARec graph which modifies how edge weights are calculated. We calculate a hand-engineered similarity between each Pull-Request in the Repository based on a number of content features. This method produces recommendations for completely unseen Pull-Requests via Personalized PageRank, and we fine-tune the teleport parameter to avoid over/under-fitting.

We also explore the effectiveness of Supervised Random Walks for link prediction between Reviewers and Pull-Requests in a bipartite graph. We engineer features from our data set and implement the SRW algorithm. Due to the size of our data set and computing limitations, we are unable to train on the entire data set, so we work with a limited portion of

the data, which likely affects the performance of our model. However, we do see some improvement in the loss function during the training, so with more computational resources it is likely that this method can produce better results.

Future research directions include building on our results here by using graph based predictions, along with graph attributes such as node degree, shortest path etc. as features in down stream supervised classifiers. It is also possible to use the node features with Graph Neural Networks for link prediction, and combining the rich edge data also available in the data set with GNNs could be an exciting application of novel research.

References

[1] L. Backstrom and J. Leskovec. "Supervised Random Walks: Predicting and Recommending Links in Social Networks". In: *arXiv:1011.4071 [physics, stat]* (Nov. 2010). arXiv: 1011.4071. URL: http://arxiv.org/abs/1011.4071 (visited on 11/06/2019).

[2] Bahman Bahmani, Abdur Chowdhury, and Ashish Goel. "Fast Incremental and Personalized PageRank". In: *arXiv:1006.2880 [cs]* (Aug. 2010). arXiv: 1006.2880. URL: http://arxiv.org/abs/1006.2880 (visited on 11/06/2019).

[3] Mukund Deshpande and George Karypis. "Item-based top-N Recommendation Algorithms". In: *ACM Trans. Inf. Syst.* 22.1 (Jan. 2004), pp. 143–177. ISSN: 1046-8188. DOI: 10.1145/963770.963776. URL: http://doi.acm.org/10.1145/963770.963776 (visited on 12/11/2019).

[4] Chantat Eksombatchai et al. "Pixie: A System for Recommending 3+ Billion Items to 200+ Million Users in Real-Time". In: *arXiv:1711.07601 [cs]* (Nov. 2017). arXiv: 1711.07601. URL: http://arxiv.org/abs/1711.07601 (visited on 11/06/2019).

[5] Georgios Gousios. "The GHTorrent dataset and tool suite". In: *Proceedings of the 10th Working Conference on Mining Software Repositories*. MSR '13. event-place: San Francisco, CA, USA. Piscataway, NJ, USA: IEEE Press, 2013, pp. 233–236. ISBN: 978-1-4673-2936-1. URL: http://dl.acm.org/citation.cfm?id=2487085.2487132.

[6] Georgios Gousios et al. "Work Practices and Challenges in Pull-based Development: The Integrator's Perspective". In: *Proceedings of the 37th International Conference on Software Engineering - Volume 1*. ICSE '15. event-place: Florence, Italy. Piscataway, NJ, USA: IEEE Press, 2015, pp. 358–368. ISBN: 978-1-4799-1934-5. URL: http://dl.acm.org/citation.cfm?id=2818754.2818800 (visited on 10/16/2019).

[7] Jing Jiang et al. "Who should comment on this pull request? Analyzing attributes for more accurate commenter recommendation in pull-based development". en. In: *Information and Software Technology* 84 (Apr. 2017), pp. 48–62. ISSN: 0950-5849. DOI: 10.1016/j.infsof.2016.10.006. URL: http://www.sciencedirect.com/science/article/pii/S095058491630283X (visited on 10/18/2019).

[8] Tsang-Wei Edward Lee and Wen-Chien Chen. "Applying Link Prediction for Repository Recommendation on GitHub". In: 2015.

[9] Manoel Limeira de Lima Júnior et al. "Developers Assignment for Analyzing Pull Requests". In: *Proceedings of the 30th Annual ACM Symposium on Applied Computing*. SAC '15. event-place: Salamanca, Spain. New York, NY, USA: ACM, 2015, pp. 1567–1572. ISBN: 978-1-4503-3196-8. DOI: 10.1145/2695664.2695884. URL: http://doi.acm.org/10.1145/2695664.2695884 (visited on 10/16/2019).

[10] Lawrence Page et al. "The PageRank Citation Ranking: Bringing Order to the Web." In: 1999.

[11] Mohammad Masudur Rahman, Chanchal K. Roy, and Jason A. Collins. "CORRECT: Code Reviewer Recommendation in GitHub Based on Cross-Project and Technology Experience". In: *2016 IEEE/ACM 38th International Conference on Software Engineering Companion (ICSE-C)*. May 2016, pp. 222–231.

[12] Patanamon Thongtanunam et al. "Who should review my code? A file location-based code-reviewer recommendation approach for Modern Code Review". In: *2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*. ISSN: 1534-5351. Mar. 2015, pp. 141–150. DOI: 10.1109/SANER.2015.7081824.

[13] Jason Tsay, Laura Dabbish, and James Herbsleb. "Influence of Social and Technical Factors for Evaluating Contribution in GitHub". In: *Proceedings of the 36th International Conference on Software Engineering*. ICSE 2014. event-place: Hyderabad, India. New York, NY, USA: ACM, 2014, pp. 356–366. ISBN: 978-1-4503-2756-5. DOI: 10.1145/2568225.2568315. URL: http://doi.acm.org/10.1145/2568225.2568315 (visited on 11/07/2019).

[14] Ximeng Wang et al. "Mixed Similarity Diffusion for Recommendation on Bipartite Networks". In: *IEEE Access* 5 (2017), pp. 21029–21038. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2017.2753818.

[15] Haochao 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)*. May 2016, pp. 29–35. DOI: 10.1109/CSI-SE.2016.013.

[16] Yue Yu et al. "Reviewer recommendation for pull-requests in GitHub: What can we learn from code review and bug assignment?" en. In: *Information and Software Technology* 74 (June 2016), pp. 204–218. ISSN: 0950-5849. DOI: 10.1016/j.infsof.2016.01.004. URL: http://www.sciencedirect.com/science/article/pii/S0950584916000069 (visited on 10/18/2019).

[17] Yue Yu et al. "Reviewer Recommender of Pull-Requests in GitHub". In: *2014 IEEE International Conference*

on Software Maintenance and Evolution. ISSN: 1063-6773. Sept. 2014, pp. 609–612. DOI: 10.1109/ICSME.2014.107.