Journal of Universal Computer Science, vol. 27, no. 11 (2021), 1240-1271
submitted: 11/2/2021, accepted: 9/8/2021, appeared: 28/11/2021 CC BY-ND 4.0

Leveraging multifaceted proximity measures among developers in predicting future collaborations to improve the social capital of software projects

Amit Kumar
(Indian Institute of Information Technology Allahabad, Prayagraj, Uttar Pradesh, India
https://orcid.org/0000-0002-2945-4300, rsi2019004@iiita.ac.in)

Sonali Agarwal
(Indian Institute of Information Technology Allahabad, Prayagraj, Uttar Pradesh, India
https://orcid.org/0000-0001-9083-5033, sonali@iiita.ac.in)

Abstract: Social capital is an asset earned by people through their social connections. One of the motivations among developers to contribute to open source development and maintenance tasks is to earn social capital. Recent studies suggest that the social capital of the project has an impact on the sustained participation of the developers in open source software (OSS). One way to improve the social capital of the project is to help the developers in connecting with their peers. However, to the best of our knowledge, there is no prior research which attempts to predict future collaborations among developers and establish the significance of these collaborations on improving the social capital at the project level. To address this research gap, in this paper, we model the past collaborations among developers on version control system (VCS) and issue tracking system (ITS) as homogeneous and heterogeneous developer social network (DSN). Along with the novel path count based features, defined on proposed heterogeneous DSN, multifaceted proximity features are used to generate a feature set for machine learning classifiers. Our experiments performed on 5 popular open source projects (Spark, Kafka, Flink, WildFly, Hibernate) indicate that the proposed approach can predict the future collaborations among developers on both the platforms i.e. VCS as well as ITS with a significant accuracy (AUROC up to 0.85 and 0.9 for VCS and ITS respectively). A generic metric- recall of gain in social capital is proposed to investigate the efficacy of these predicted collaborations in improving the social capital of the project. We also concretised this metric on various measures of social capital and found that collaborations predicted by our approach have significant potential to improve the social capital at project level (e.g. Recall of gain in cohesion index up to 0.98 and Recall of gain in average godfather index up to 0.99 for VCS). We also showed that structure of collaboration network has an impact on the accuracy and usefulness of predicted collaborations. Since the past research suggests that many newcomers abandon the open source project due to social barriers which they face after joining the project, our research outcomes can be used to build the recommendation systems which might help to retain such developers by improving their social ties based on similar skills/interests.

Keywords: Developer Social Network (DSN), Online Collaboration, Social Capital, Link Prediction, Heterogeneous Developer Social Network, Homogeneous Developer Social Network, Issue Tracking System, Version Control System
Categories: D.2, D2.7, H.3.4, H.3.5, H.4, M.3
DOI: 10.3897/jucs.76602

1 Introduction

Bug fixing is largely a collaborative task where developers collaborate to fix the issues or bugs. Two major information systems where developers contribute and discuss towards fixing the issues are issue tracking system (ITS) and version control system (VCS). The discussion among developers on issue tracking system and their commit activities in version control systems lead to form an implicit DSN. For instance, the interaction among developers on ITS can be modelled as the network where developers are nodes and edges between pair of developers signify the fact that they have commented on at least one common bug report. Similarly, the DSN formed out of version control data could have nodes as authors and edges between pair of nodes can signify that they have committed at least one common file.
DSNs have been investigated in the past for solving many software engineering problems. It has been used for defect prediction [Meneely et al., 2008] [Bird et al., 2009] [Abreu and Premraj, 2009], automatic bug triaging [Wu et al., 2011] [Bhattacharya and Neamtiu, 2010] [Banitaan and Alenezi, 2013] [Jeong et al., 2009], for code review [Kerzazi and El Asri, 2016] and evolution of software and developer communication patterns [Bhattacharya et al., 2012] [Datta et al., 2011] [Hong et al., 2011] [Kumar and Gupta, 2013] etc.

One of the motivations for OSS contributors to contribute to the development and maintenance of the open source software is to improve their social connections and social status in the open source development community. In management and social science, the rewards obtained by individuals owing to their position in community and connections with their peers are referred to as social capital. Though social capital has been very popular in social science [Burt, 1998] [Chiu et al., 2006] [Aguilera, 2002] [Brown and Ferris, 2007] [Guiso et al., 2004] [Hahn et al., 2008], a systematic classification of social capital and network metrics to measure it, was proposed only recently by Jackson [Jackson, 2019].

In open source software development, the importance and impact of social capital has been studied from team composition [Tan et al., 2007], return from investment [Méndez-Durón and García, 2009] and project's success point of view [Chou and He, 2011] [Singh et al., 2011] [Daniel et al., 2018]. Recently Qui et al. [Qiu et al., 2019] found that social capital of developers has a significant impact on their sustainability with the project. One way to help the developers in improving their social capital is by recommending new peers whom they can collaborate with, towards completing the given task e.g. bug fixing.

In this paper, we use link prediction approach in recommending the new ties among developers to improve their social capital. Link prediction is a popular problem and has been explored extensively in the field of data mining and social network analysis. Hasan and Jaki [Al Hasan and Zaki, 2011], Martinez et al. [Martínez et al., 2016] and Samad et al. [Samad et al., 2020] provide a exhaustive literature review on this area. Some past studies have utilized link prediction approach in solving some of the software engineering problems. For instance, software engineering researchers have used link prediction in detecting architecture smells [Díaz-Pace et al., 2018], recommending right project or repository to Github developers [Nielek et al., 2016] [Akula et al., 2019], predicting design dependencies [Diaz-Pace et al., 2018], predicting right web service to the web service developers [Huang et al., 2013] etc.

Compared with the past research on link prediction in software engineering, our work is novel in many aspects as indicated below:

1. First, we predict the future collaborations among the developers based on their

current associations and activities. That is, we predict pairs of developers who have not communicated in the past, but are likely to communicate in the future to fix a bug. This can be helpful to design the tools that can recommend developers who are likely to cooperate in a bug fix and expedite the bug fixing process. This is a useful problem also because its solution can be useful in improving the social capital of developers in the project which in turn results into the longer associations of the developers with the project as described by Qiu et al. [Qiu et al., 2019]. To the best of our knowledge, no prior study has attempted the link prediction approach in predicting future collaborations among developers in open source projects.

2. Second, instead of just restricting ourselves to the homogeneous representation of DSN like many past studies, we proposed an integrated heterogeneous network schema and defined various meta paths based on this schema. Along with some activity based features, we used these meta paths to enrich the features set for our classifiers. In this way, we use three kinds of features i.e. based on homogeneous DSN, based on heterogeneous DSN and based on the number of activities of developers on VCS and ITS platform, making our feature set diverse enough. While in homogeneous DSN, nodes and edges are of same types, in heterogeneous DSN, both nodes and edges can be of different types. For example, we can have Homogeneous DSN where nodes are developers and edge between developers represent the fact that they commented on the same bug report at least once. Heterogeneous DSN helps to capture richer information present in software bug (We use Bug and issue interchangeably in this article) repositories. The developers who have commented on two different bug reports of the same component can also be connected through a path in heterogeneous DSN even if they did not comment on the same bug report. Such deeper multiple types of connections can not be leveraged using homogeneous DSN. Despite being very popular in the data mining field, no past study in software engineering domain leverages heterogeneous network structure for link prediction task, to the best of our knowledge.

3. Third, we relied on two sources of data for the construction of the homogeneous and heterogeneous networks i.e. issue comments and commit history of the projects. That is, we also consider developers committing the same file as collaborators. In this way, we have two versions of homogeneous as well as heterogeneous networks, one for comment data and another for commit data.

4. Finally, apart from using common metrics, widely used to measure the performance of link prediction tasks e.g Precision, Recall, AUROC, we also investigated the usefulness and relevance of new links predicted by our method. Apart from using various standard network measures described recently by Jackson [Jackson, 2019], we proposed two novel network measures to investigate how much gain in the average social capital of the developers working in open source software project could be achieved using our method. We also investigate if different levels of collaborations (collaboration based on commit activity and collaboration based on comment activities) among developers have different potential of improving their social capital.

In nutshell, we investigate the following research questions in this paper.

RQ1: Can we use link prediction technique to predict the future collaborations among the developers? How accurately the popular binary classifiers can predict the future collaborations in the bug fixing process?

RQ2: How useful the recommended developer-developer links are in terms of improving the social capital of the developers?

RQ3: How does the performance of link prediction vary across different types of DSNs in improving the social capital i.e. comment data based DSN or commit data based DSN? What causes this difference in performance?

To address the above research questions, 5 popular open source projects i.e. Spark, Flink, Hibernate, Kafka and Wildfly have been used. We found that our approach could improve the social capital of the developers significantly. It is also noted that the links predicted using commit based DSN has more potential to improve the social capital of the developers.

The rest of the paper is organized as follows- section 2 provides the overview of related work to our paper, section 3 discusses the methodology, section 4 presents the experimental setup and details of the dataset, we used while section 5 discusses the results. Finally, we conclude the paper in section 6.

2 Related Work

An implicit developer social network is created when developers work on same artefact to perform their development and maintenance duties. Here we review some of the most related work on DSN and how they have been used to solve some interesting problems in software engineering. For example, Canfora et al. [Canfora et al., 2012] identify the mentors to the newcomer developers by analyzing the mailing lists of the project. Bird et al. [Bird et al., 2006] performed network analysis on DSN constructed from mailing lists to discover the social status of OSS contributors in the network and how this status is related to their commit activities. Hong et al. [Hong et al., 2011] compared the evolution of general social network e.g. Facebook, Twitter etc. with the evolution of developer social network. Kumar and Gupta [Kumar and Gupta, 2013] on the other hand, extend the work of Hong et al. by investigating the evolution of DSN for other OSS projects and how the structural properties of these DSNs influence the efficiency of bug fixing process. Zhang et al. [Zhang et al., 2013] [Zhang et al., 2016] used heterogeneous DSN constructed from ITS data to study the nature of developers contributions and to automate the bug triaging process. We also use DSN for our study. However, our study is more comprehensive as we use three different types of DSNs-commit based Homo-DSN, comment based Homo-DSN and integrated Hetero-DSN. Moreover the problem which we attempt to solve is significantly different from the past research.

Developers having many social connections hold social capital and always desirable in an organization. Many studies including Hahn et al. [Hahn et al., 2008] shows how prior relationships among developers can help in team formation in the project. Newcomer software developers usually prefer to be part of software teams where they know some of their peers [Casalnuovo et al., 2015]. Mendez et al. [Mendez et al., 2018] report that newcomer software developers face social and technical barriers when they join the open source community and how tools can help them to address this issue. Qiu et al. took these studies forward to see how social capital at the project level has an impact on the sustained participation of the developers in the OSS project. We complement these studies on social capital firstly by introducing the concrete network measures to evaluate the social capital of open source project as proposed by jackson [Jackson, 2019] and Qiu et al. [Qiu et al., 2019]. Secondly, the importance of link prediction on various types of DSNs to improve social capital is also demonstrated.

3 Methodology

In this section, we cover some background concepts related to our work and discuss how they can be leveraged in answering our research questions.

3.1 Homogeneous DSN

Both, VCS as well as ITS have been used to construct the DSN in the past. In homogeneous DSN, the nodes and edges are of the same types i.e. nodes are developers and edges between pair of developers signify the communication/collaboration that happened between pair of developers. To further illustrate the Homo-DSN, let us consider the toy example of ITS and VCS data shown in Table 1 and Table 2. Table 1 shows the issue comment data where developers discuss the issue by making comments on ITS.

Table 1: Toy Example of Issue Comment Data

CommentID	IssueID	Developer	Reporter	Assignee	Component
CMT_1	I_1	D_1	R_1	A_1	COM_1
CMT_3	I_2	D_3	R_1	A_2	COM_2
CMT_4	I_2	D_2	R_1	A_2	COM_2
CMT_5	I_2	D_4	R_1	A_2	COM_2
CMT_6	I_3	D_4	R_2	A_2	COM_1
CMT_7	I_3	D_5	R_2	A_2	COM_1

Table 2: Toy Example of Commit Data for issues mentioned in Table 1

IssueID	CommitID	Developer	File Path	Package
I_1	CIT_1	D_1	$\alpha_1\|\beta_1\|\gamma_1\|\delta_1.py$	$\alpha_1\|\beta_1\|\gamma_1$
I_1	CIT_2	D_5	$\alpha_1\|\beta_1\|\gamma_2\|\delta_2.py$	$\alpha_1\|\beta_1\|\gamma_2$
I_2	CIT_3	D_6	$\alpha_1\|\beta_1\|\gamma_1\|\delta_1.py$	$\alpha_1\|\beta_1\|\gamma_1$
I_3	CIT_4	D_5	$\alpha_1\|\beta_2\|\gamma_3\|\delta_3.py$	$\alpha_1\|\beta_2\|\gamma_3$
I_1	CIT_5	D_6	$\alpha_1\|\beta_1\|\gamma_2\|\delta_2.py$	$\alpha_1\|\beta_1\|\gamma_2$
I_2	CIT_6	D_3	$\alpha_1\|\beta_1\|\gamma_1\|\delta_4.py$	$\alpha_1\|\beta_1\|\gamma_1$

The corresponding Homo-DSN (graph on the left side of the Figure 1) has developers as nodes and edge exist between the pair of developers if they have commented on at least one common issue. Similarly Table 2 shows the commit activities corresponding to the issues mentioned in Table 1. The commit based Homo-DSN is shown on the right side of Figure 1.

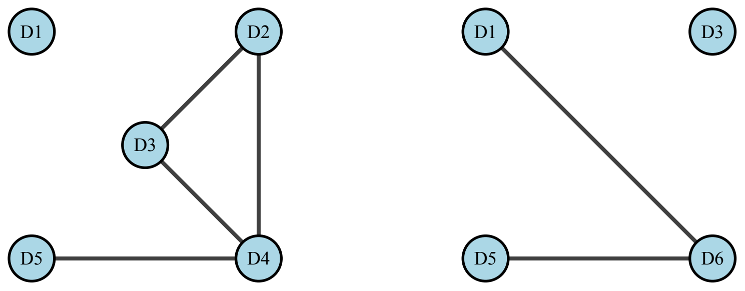

Figure 1: Homogeneous Developer Social Networks: Based on Table1(Left) and Based on Table 2(Right)

(D_1, D_6) and (D_5, D_6) are only two pairs where developers have committed on a common file and hence this network is relatively sparser. The network proximity measures e.g. common neighbours between developers implies how closer they are in their work profiles.

3.2 Proposed integrated heterogeneous DSN Schema

While computing the proximity measures between developers, Homo-DSN have some limitations. For instance, two developers commenting on two different issue reports of the same software component or developers commenting on two different issue reports which have caused to change the same file, are semantically closer with each other but such similarity might be overlooked in Homo-DSN. To address this issue, we proposed Hetero-DSN, based on the network schema shown in Figure 2.

It can be noted that Figure 2 shows the meta-network and not the actual network. The actual Hetero-DSN is instantiated based on this abstract network schema. The nodes of the Hetero-DSN are connected by following the meta paths. A meta path is a path defined over an abstract network schema. It is the sequence of node types where node types are connected through edge types. For instance, in figure 2, a meta path, developer-commit-file-directory exists between developer and directory. Needless to say, there are many possible meta paths that can be defined on meta-network (network schema). However, only a few, which are relevant to our problem have been used in our study. Since we require to measure the proximity among developers, all our meta paths have developers as end nodes in the meta path. The intuition behind identifying the meta paths is that developers who share some similarities in their work profiles (e.g. developers who have worked on different issues, found in the same component) are likely to work together in future even if they have not worked together in the past. All the meta paths used in our work is listed in Table 3.3.2 and Table 3.3.2. Each meta path signifies a different semantic meaning and captures the different hidden similarities between the developers. For example, using the meta path D-I-A-I-D, developers commenting on two different issue reports, assigned to the same assignee could be connected. It can be noted

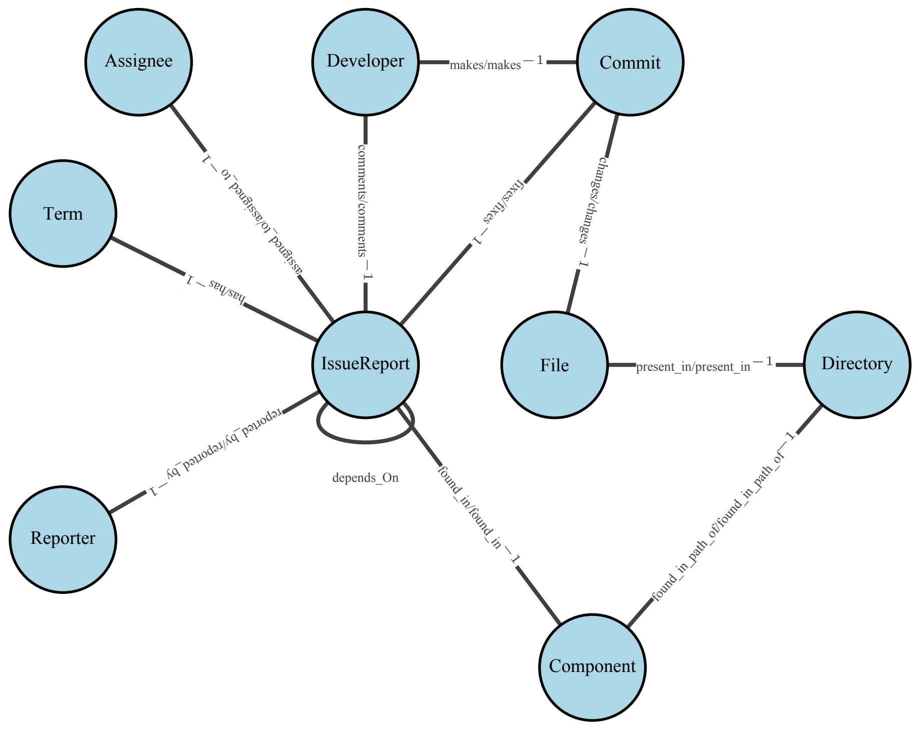

Figure 2: Schema of Heterogeneous Developer Social Network

that developers who are not connected in Homo-DSN can be connected in Hetero-DSN. For instance, in Homo-DSN based on comment data, (D_1, D_5) and (D_1, D_2) are not connected. However, both of them could be connected in Hetro-DSN with the help of paths, D_1–I_3–COM_1–I_3–D_5 (using meta path D-I-C-I-D) and D_1–I_1–R_1–I_1–D_2 (using meta path D-I-R-I-D) respectively. Similarly in Homo-DSN based on commit data, (D_1, D_3) and (D_3, D_5) are not connected to each other while they are connected in Hetero-DSN using the paths D_1—CIT_1—$\alpha_1|\beta_1|\gamma_1|\delta_1.py$—$\alpha_1|\beta_1|\gamma_1$—$\alpha_1|\beta_1|\gamma_1|\delta_4.py$—$CIT_6$—$D_3$ (using meta path D-CIT-F-DR-F-CIT-D) and D_3–CIT_6–I_2–A_2–I_3–CIT_4–D_5 (using meta path D-CIT-I-A-I-CIT-D) respectively. From the above examples, it is clear that a meta path describes the way to connect concrete objects and hence there can exist multiple concrete paths following a specific meta path. The proximity among developers can be measured using Homo-DSN based measures (e.g. common neighbours) and Hetero-DSN based measures (e.g. path counts based on various meta paths). Good proximity scores among developers indicate the similarity in their profiles and developers who share good proximity measures are likely to collaborate in the bug fixing process. In our work, we use Homo-DSN and Hetero-DSN based proximity measures to build supervised learning classifiers in predicting future collaborations among them. We constructed three DSNs in total-Homo-DSN based on comment data, Homo-DSN based on commit data and one integrated Hetero-DSN based on Meta-Network shown in Figure 2.

3.3 Link Prediction

In link prediction, given a network in time t, future connections are predicted among nodes of the network in time t+1 [Liben-Nowell and Kleinberg, 2007]. Predicting future collaborations among existing developers is equivalent to predicting the new edges (links) in the DSN. Note that we make predictions of future links among developers who are common across times t and t+1. Even though we do not predict possible links with new developers entering the project, We found a significant number of new connections between those nodes that are common across the time periods (t and t+1). These links can be predicted by our approach. We create three DSNs for our study-comment based Homo-DSN, commit based Homo-DSN and Hetero-DSN based on the schema defined in Figure 2. To predict new collaborations at the comment and commit level we apply link prediction on comment based Homo-DSN and commit based Homo-DSN respectively. It should be noted that though we do not apply link prediction on Hetero-DSN, the path based measures defined on it are used as features in classifiers used for both types of Homo-DSNs.

We used three types of features in our link prediction classifiers i.e. features based on homogeneous DSN, features based on heterogeneous DSN and features based on the number of contributions of the developers. In total, we used 16 features for both commits based and comment based DSNs. Among these 16 features, 5 are based on homogeneous DSN, 5 are contribution based features and 6 are based on the path counts in the heterogeneous DSN. Details of these features are given as follows.

3.3.1 Features based on homogeneous DSN

Since we studied the DSNs at two levels i.e. comment level and commit level, we constructed two different DSNs and computed following measures based on both the types of DSNs.

3.3.1.1 ***Number of common neighbours :***

The intuition behind this metric is that if two developers sharing more number of neighbours between them, then they themselves are likely to be connected in the future. Adamic Adar, Jaccard Coefficient and Preferential Attachment measures are also based on the number of neighbours of the developers and are defined below.

3.3.1.2 ***Jacard Coefficient:***

This is defined as the number of common neighbours divided by the total number of neighbours of both the developers. Mathematically, Jaccard Coefficient (JC) is defined as

$$JC\,(d1, d2)=\frac{|N(d1)\cap N(d2)|}{|N(d1)\cup N(d2)|}$$

where $N\,(d)$ is the set of neighbours of developer d.

3.3.1.3 Adamic/Adar:

Adamic/Adar (AA) score calculation gives less significance to those common neighbours who themselves have a very large number of neighbours/connections. It is defined as follows

$$AA\,(d1, d2){=}\sum\nolimits_{D\in N(d1)\cap N(d2)} \frac{1}{\log|N(D)|}$$

3.3.1.4 Preferential Attachment:

The Preferential Attachment (PA) score is simply the product of the number of neighbours of two developers. It is defined as follows

$$PA\,(d1, d2){=}|N\,(d1)\,|\cdot|N\,(d2)\,|$$

3.3.1.5 Length of Shortest Path:

This measure returns the length of the shortest path between pair of developers in the graph. Shorter the length between two nodes in the graph, higher are the chances that they will collaborate in future.

3.3.2 Features based on heterogeneous DSN:

Hetero-DSN based features are prepared based on meta paths defined on Hetero-DSN. For predicting future links in comment based Homo-DSN, we compute the number of paths i.e. path counts based on each meta path listed in Table 3.3.2 while to predict the links in commit based Homo-DSN, we use path counts based on the meta paths listed in Table 3.3.2. This way, we use 6 features based on Hetero-DSN for each of the Homo-DSNs (i.e. comment based DSN and commit based DSN).

Table 3: Meta Paths used to connect developers in Heterogeneous Developer Social Network(Based on Comment data)

D:Developer, I:Issue, A:Assignee, CIT:Commit, F:File, DR:Directory, C:Component, R:Reporter

Meta Path	**Elucidation of Meta Path**
D-I-A-I-D	Developers commenting on the issues assigned to same assignee
D-I-CIT-F-CIT-I-D	Developers commenting on issues which cause changing the same file
D-I-CIT-F-DR-F-CIT-I-D	Developers commenting on issues which cause changing files in same directory
D-I-C-I-D	Developers commenting on issues found in same component
D-I-I-D	Developers commenting on two related issues
D-I-R-I-D	Developers commenting on issues reported by same reporter

Table 4: Meta Paths used to connect developers in Heterogeneous Developer Social Network (Based on Commit data)

D:Developer, I:Issue, A:Assignee, CIT:Commit, F:File, DR:Directory, C:Component, R:Reporter

Meta Path	Elucidation of Meta Path
D-CIT-F-C-F-CIT-D	Developers authoring the files which share a component present in their directory path
D-CIT-F-DR-F-CIT-D	Developers authoring the files present in the same directory
D-CIT-I-A-I-CIT-D	Developers contributing in fixing the issues assigned to the same assignee
D-CIT-I-C-I-CIT-D	Developers contributing in fixing the issues found in the same component
D-CIT-I-I-CIT-D	Developers contributing in fixing two related issues
D-CIT-I-R-I-CIT-D	Developers contributing in fixing the issues reported by same reporter

3.3.3 Contribution based features

These features are based on the contributions of the developers, they make on the issue tracking systems and version control systems to resolve the issues. We use many similar but distinguishable measures to compute the number of contributions of the pair of developers on ITS and VCS. Our intuition is that developers who are more active and contribute frequently are more likely to form new associations with other developers. For predicting the links in comment based DSN, we use contributions made to ITS while predicting links in commit based DSN, we use contributions made on VCS. We use the following contribution based measures as features in our link prediction classifiers.

*3.3.3.1 **Sum of the number of bug reports/files, the developers worked upon :***

For comment based DSN, we compute the sum of the number of bug reports, the two developers commented upon while for commit based DSN, we compute the sum of files the pair of developers committed in VCS.

*3.3.3.2 **Sum of the number of comments/commits, made by the developers :***

For comment based DSN, we compute the sum of the number of comments made by two developers while for commit based DSN, we compute the sum of commits made by the pair of developers in VCS.

*3.3.3.3 **Sum of the number of distinct components, the developers worked upon :***

This feature measures the expertise breadth of a developer. If a developer has breadth in expertise and comments on bug reports associated with many components (or commits files to fix the issues associated with many components), it suggests that he/she has diversity in his/her expertise.

*3.3.3.4 **Sum of the neighbours of the developers :***

The intuition behind choosing this feature is that if developers are directly engaged with many developers in the discussion for a bug fix, then they are more likely to form newer connections. In other words, developers who communicate/collaborate more with other developers are more likely to form new ties. We compute the sum of neighbours of pair of developers in both commit based and comment based DSNs.

*3.3.3.5 **Shared TFIDF score :***

The intuition behind this measure is that the developers who work on the issue report with similar text are likely to work together. In this measure, we extract the keywords from the title and description of the issue reports which have been worked upon by the pair of developers. Then we compute the TF-IDF based cosine similarity score [Manning et al., 2008] between sets of keywords fetched from issue reports of developer pairs.

3.4 Social capital

The social capital has a significant impact on the success of the Open Source Software (OSS) project and longer association of contributors with the project [Daniel et al., 2018] [Qiu et al., 2019].

To see the importance of our predicted links in improving the social capital of the developers in the project, we require some concrete measures to measure the social capital. Recently, Jackson [Jackson, 2019] proposed some network measures for measuring social capital. However, these metrics are defined at individual level while we require some project level measures. Qui et al. [Qiu et al., 2019] proposed some project level metrics, but we found some drawbacks in their approach and did not use them. Qui et al. consider two developers familiar if they have worked on the same project. In a large open source project, this assumption may not be true. Since developers introduce less number of defects if they are focused in their code contributions [Posnett et al., 2013], OSS strive for focused contributions from the developers and hence the assumption made by Qui et al. is not always true. That is, two developers who are part of the same project may not share their expertise and can be quite unaware of each other's contributions. We observed such a phenomenon in the projects of our case study also. Some of the developers are confined to only a few source code packages and source files. To address these issues, first, three main individual measures defined by Jackson are converted to the project level measures by taking the average value of the measure. The higher value of this metric indicates the higher level of overall social capital of the project. Second, we propose two new project level metrics i.e. cohesion index and average agility index. Since we compute the values of metrics based on developers' interactions at the file level and issue level, the drawbacks present in the metrics proposed by Qui et al are no longer present in our metrics. In total, we used 5 metrics which are described below.

3.4.1 Cohesion Index (CI):

Cohesion Index measures the level of direct collaboration existing among the developers of the project. Higher the value of this metric, higher is the pairwise familiarity among the developers. Mathematically, it can be defined as follows:

$$CI{=}\frac{2}{n(n-1)}\left(\sum_{x>y} e_{xy}\right)$$

where e_{xy} is 1 if (x, y) are adjacent to each each other and 0 otherwise. n is the number of nodes in the network.

3.4.2 Average Decay Centrality (ADC):

The Decay Centrality of a node is high if a node can reach to many other nodes. In a network, nodes can be reached directly (adjacent nodes) or through a path. The farther reachable nodes contribute less in the computation of the decay centrality in comparison to those who are closer to the node. Since we measure social capital at the project level and not at the developer level, we take the average of the decay centralities of all the developer nodes in the project. Mathematically, it is defined as follows:

$$ADC = \frac{1}{n}\left(\sum_x \sum_{q=1}^{m} \alpha^q \left|\chi_x^q\left(\mathbf{G}\right)\right|\right)$$

Here , $\left|\chi_x^q\left(\mathbf{G}\right)\right|$ denotes the number of nodes reachable from x , in graph G , using q hops. α is a decay factor and m is the maximum number of hops considered in the computation. n is the total number of nodes (developers) in the graph. In our calculation, we took the value of α as 0.5 and value of m as 10, since, in all the practical networks, it is rare that two nodes are only connected with more than 10 hops.

3.4.3 Average Godfather Index (AGI):

A Godfather Index of a node in network describes its ability to connect pair of nodes which are otherwise not connected. This measures the bridging ability of a node. A graph having good value of AGI indicates that it has many Godfather nodes. Mathematically, the AGI is defined as follows:

$$AGI{=}\frac{1}{n}\left(\sum_x \sum_{\substack{z>y: \\ e_{zy}=e_{yz}=0}} e_{zx}e_{yx}\right)$$

Where, e_{ab} is 1 if nodes a and b are adjacent to each other and 0 otherwise. The term $\sum_{\substack{z>y: \\ e_{zy}=e_{yz}=0}} e_{zx}e_{yx}$ indicates the Godfather index of individual node x.

3.4.4 Average Diffusion Index (ADI):

The Diffusion Index of a node in the network is its ability to diffuse the information to other nodes. It should be noted that though Diffusion Index and Decay Centrality sound similar, they are quite distinct. While Decay Centrality of a node depends on the node counts which are reachable from the given node, Diffusion Index also counts the number of paths which exist to that reachable set of nodes. This measure is computed based on the $n \times n$ probability matrix α where α_{xy} indicates the probability of passing the information from x to y. We choose the default probability as 0.5 (to make a case of equally likely) and hence (x, y) entry in our matrix is set as 0.5 if edge (x, y) exist in the graph, otherwise, the value is set as 0. The term $\left[\alpha^q\right]_{xy}$ indicates the (x, y) entry in the q^{th} power of the matrix α. Average Diffusion Index is defined as follows:

$ADI = \frac{1}{n}\left(\sum_x \sum_y \sum_{q=1}^{m} [\alpha^q]_{xy}\right)$.

Here, the term $\sum_y \sum_{q=1}^{m} [\alpha^q]_{xy}$ indicates the diffusion index of node x. The value of m chosen by us is 10 as it is very unlikely for a node to be connected via path length of more than 10 if it is not connected through a path of length ≤ 10.

3.4.5 Average Agility Index (AAI):

The intuition behind this metric is that a developer can expect a more agile response from those developers who are closer to him in the collaboration network. The high value of AAI indicates that developers in the projects are reachable to each other via a short path. Mathematically, it can be defined as follows:

$AAI = \frac{1}{n}\sum_x \left(\frac{n-1}{\sum_{y=1}^{n-1} MinDist(y,x)}\right)$, where, $MinDist\,(y,x)$ is the minimum distance between x and y and n is the total number of nodes in the network.

3.5 Measuring the improvement in the social capital:

Developers increase their associations over time and hence improve their social capital. Here we investigate if we can help them to improve their social capital by recommending the links in developer social network. In link prediction, we use three time stamps-t_1, t_2 and t_3. First, we identify the common developers between two time periods i.e. (t_1, t_2) and $[t_2, t_3)$. Then features extracted based on data between t_1 and t_2 are used to train the classifier. This classifier is used to predict the links among common developers during $[t_2, t_3)$. To evaluate our method, we make use of three DSNs-The original DSN N_O constructed from data during (t_1, t_2), the DSN N_A is constructed by augmenting N_O with the actual new links formed among the common developers during $[t_2, t_3)$ and a DSN N_P which is constructed by augmenting N_O with all the new predicted links among the common set of developers during $[t_2, t_3)$ by our classifiers. We compute the values of various social capital measures defined above for these three networks and use Recall of Gain in Social Capital (RGSC) to evaluate our method. Mathematically, we can define it as follows:

$$RGSC = \frac{\text{Social Capital of } N_P \text{ —Social Capital of } N_O}{\text{Social Capital of } N_A \text{ —Social Capital of } N_O}$$

RGSC describes the fraction of the additional social capital that could be earned at time t_2, by a possible recommender system built upon our approach, which would otherwise be possible to achieve at t_3.

Based on the measures defined for social capital in the previous section, we use 5 different concrete measures for RGSC i.e. Recall of Gain in AAI (RGAAI), Recall of Gain in ADI (RGADI), Recall of Gain in AGI (RGAGI), Recall of Gain in ADC (RGADC) and Recall of Gain in CI (RGCI).

4 Experimental setup and dataset

In this section, First, we explain dataset and classifiers used in our works. Then we propose our experimental design with pseudo-code describing our process of answering our research questions.

4.1 Dataset and classifiers:

To perform our study, we used ITS and VCS data of 5 Open Source projects (Spark, Kafka, Flink, Hibernate and Wildfly) from the openly available SEOSS dataset [Rath and Mäder, 2019]. We chose SEOSS dataset because it has an integrated view of ITS and VCS data. That is, issues and their related commits are linked together. We chose these project as they are good mix of different types of categories. While Spark, Kafka and Flink are popular projects in big data domain and vary in their nature and size, Wildfly is a popular application server software and Hibernate is popular object relational mapping software. The dataset supports the generality of our study. One common issue in using ITS and VCS data together is that developers have multiple identities in VCS and ITS and it is not easy to merge these identities. Though many automatic identity merging approaches have been proposed by researchers recently [Amreen et al., 2020], to avoid the inherent potential errors in the process, we performed the identity merging task manually and hence our data is free from ambiguity related to multiple identities of the developers. The details of our dataset are given in Table 5. For classification, we used Weka [Hall et al., 2009] implementation of 6 popular classifiers-J48 (Decision Tree), Random Forest, Logistic Regression, Logistic Model Tree, SMO (Support Vector Machine) and Bayes Net.

Table 5: Details of the Dataset

Project	#Issues	#Comments	#Commenters	#Commits	#Files	#Authors
Flink	4810	48783	480	4354	10629	288
Hibernate	1871	6050	894	1793	6904	182
Kafka	3484	15766	966	2777	2938	319
Spark	11797	41779	2816	7189	7056	657
WildFly	2473	6919	524	8791	10800	164

4.2 Experimental design:

To answer our research question 1, we use supervised link prediction approach as proposed by Hasan et al. [Al Hasan et al., 2006]. The supervised link prediction problem can be posed as a binary classification problem. We prepare the classification dataset by splitting the 2 years of ITS and VCS data of projects into training (Nov 2015-Oct, 2016) and testing (Nov 2016-Oct, 2017) time periods. To create the data for classifiers, we consider developers who are common across the periods, since we are predicting new links that might emerge among the developers who are present in the training period as well as a testing period. We need to label this dataset before running the classifier. To do so, we identify all those developers who did not share an edge in the training period. We then check whether the pair shared a link in the test period, if yes it is labelled as 1 (positive), otherwise it is labelled as 0 (negative). Typically all link prediction tasks face a problem of class imbalance where the instances with negative labels outnumber the instances with positive labels. To address this issue, we randomly chose as many negative instances (from the entire pool of negative instances) as positive instances to make the dataset

Conventions and assumptions:
P: Set of projects for our study = {Spark,Wildfly,Flink,Kafka ,Hibernate}
C: Set of all classifiers we used for prediction tasks = {SVM,BayesNet,J48,Logistic Model Tree, Logistic Regression,Random Forest}
D: Kinds of dataset = {commit,comment}
coll(x,y,t): This function returns 1 if developer x and y collaborated in time period t, otherwise 0 is returned. t can be either a training period or test period. Commenting on same issue report and committing the same source file are considered collaborations for comment data and commit data respectively.
min(A, B): The function receives two sets as arguments and returns the one with minimum cardinality.
max(A, B): The function receives two sets as arguments and returns the one with maximum cardinality.
ran(A,n): The function receives a set A and a number n as arguments and returns a set with n elements randomly chosen from A.
generate_features(d): The function generates the values of all the similarity features between developer pair d.
classifier(data,10): classifier is run on the data with 10 fold cross-validation scheme and generate the performance vector [Precision, Recall, F-Measure, AUROC] and label map $M{:}Q \rightarrow L$ is the output map returned by the classifier. Here, Q is a set of developer pairs in data and L is the label set {0,1}.
find_label(L,d): This finds a label for developer pair, d in label map L.
social_capital(N): This function returns a vector([CI,AAI,ADC,AGI,ADI]) containing all the network based social capital measures for network N , defined for the project.
compute_network_measures(N): This function returns the vector ([Number of connected components, Density, Average Degree, Average Clustering Coefficient]) of numerical values of global network properties for network N.

Figure 3: Conventions and implicit functions used in our pseudo code

balanced. This is a common technique used in past research also [Al Hasan et al., 2006]. To investigate the performance of classifiers, we use 10 fold cross-validation scheme. The evaluation metrics we used for our experiments are Precision, Recall, F-Measure and AUROC (Area Under ROC curve). These are very popular metrics and have been used in link prediction literature [Liben-Nowell and Kleinberg, 2007] [Al Hasan et al., 2006] [Sun et al., 2011]. The proposed approach is presented in form of a pseudo code. Figure 3 describes the conventions and implicit/primitive functions, used in our pseudo code. The pseudo-code starts with some common operations e.g. network construction etc. which are used for the computations required to answer all RQs. This set of initialization operations are presented in Figure 4. The pseudo-code corresponding to RQ1 is presented in Figure 5. Once all future collaborations are predicted, we use the scheme presented in section 3.5 to answer our research question 2. We compute the value of RGSC with

```
∀project in P ∧ ∀classifier in C ∧ ∀dataset in D
    classifier_data = $\phi$
    $D_{train}$=The set of developers in training period
    $D_{test}$=The set of developers in test period
    $D_{common}$=$D_{train} \cap D_{test}$
    $N^{O}_{train}$=$\{(x,y) \mid (x,y) \in D_{train} \times D_{train} \wedge$ coll(x,y,train)=1$\}$
    $N^{O}_{test}$=$\{(x,y) \mid (x,y) \in D_{test} \times D_{test} \wedge$ coll(x,y,test)=1$\}$
    $PS$=$\{(x,y) \mid (x,y) \in D_{common} \times D_{common} \wedge (x,y) \notin N^{O}_{train} \wedge (x,y) \in N^{O}_{test}\}$
    $NS$=$\{(x,y) \mid (x,y) \in D_{common} \times D_{common} \wedge (x,y) \notin N^{O}_{train} \wedge (x,y) \notin N^{O}_{test}\}$
    $CDS$=$min(PS,NS) \cap ran(max(PS,NS), |min(PS,NS)|)$
```

Figure 4: Declarations and initialization used for our pseudo code

```
    ∀d in CDS:
        FV = generate_features($d$)
        $L$=$1, if\ d \in PS$, 0 otherwise
        $classifier\_data$=$classifier\_data \cup < d, FV, L >$
    $performance\_vector, label\_vector$=$classifier(classifier\_data, 10)$
    $print(performance\_vector)$    /*Answer to the RQ-1*/
```

Figure 5: Pseudo code for RQ1

respect to each of the social capital measures described in section 3.4 and report the results. The pseudo-code corresponding to RQ2 is presented in Figure 6. Finally, to answer our research question 3, we study the network properties of comment based and commit based DSN, to see if network structure of DSN can explain the difference in the performance of link prediction classifiers and their ability to improve the social capital. For this, we used some simple but relevant network metrics i.e. Number of connected components (NCC), Average Clustering Coefficient (ACC), Average Degree and Density of the graph. To compute the average clustering coefficient, we used the definition proposed by Kaiser [Kaiser, 2008] as this has been used in many past studies. The other metrics are very simple and common in graph theory. These metrics measure the level of connectedness and group structure in the network. For instance, a lower value of NCC (number of connected components) indicates that individuals in the network are well connected. Similarly higher value of ACC, Average Degree and Density also indicates cohesiveness and good connectivity among nodes (individuals). Figure 7 describes the pseudo-code capturing the steps to be executed in answering RQ3.

```
N$^{P}$=$N^{O}_{train}$
$\forall d$ in CDS:
  $if\left(find\_label\left(label\_vector,d\right)==1\right):$
    $N^{P}=N^{P}\cup\{d\}$
$N^{A}=N^{O}_{train}\cup\{(x,y)\mid(x,y)\in D_{common}\times D_{common}\wedge(x,y)\in N^{O}_{test}\}$
$OSC=social\_capital\left(N^{O}_{train}\right)$
$PASC=social\_capital\left(N^{P}\right)$
$AASC=social\_capital\left(N^{A}\right)$
for i in range(0, OSC.length):
  RGSC[i] = $\frac{\text{PASC[i] —OSC[i]}}{\text{AASC[i] —OSC[i]}}$
$print\left(RGSC\right)$                /* Answer to the RQ-2*/
```

Figure 6: Pseudo code for RQ2

```
if(dataset==’commit’):
  commit_based_measures = compute_network_measures$\left(N^{O}_{train}\right)$
$else$:
  $comment\_based\_measures=compute\_network\_measures\left(N^{O}_{train}\right)$
$print\left(comment\_based\_measures\right)$
$print\left(commit\_based\_measures\right)$          /* Answer to the RQ-3*/
```

Figure 7: Pseudo code for RQ3

5 Results and discussion:

The results are discussed with respect to the research questions posed in section 1. (RQs).

RQ1: Can we use link prediction technique to predict the future collaborations among the developers? How accurately the popular binary classifiers can predict the future collaborations in the bug fixing process?

To answer this research question, we ran multiple popular classifiers on comment based DSN and commit based DSN. The results are shown in Table 6 and Table 7. For link prediction and related tasks, AUROC has been used as the main metric to study the performance of the classifiers and associated recommender systems in past research. Hence, the analysis of results is mainly based on AUROC. From the tables, it is clear that classifiers are able to predict the future collaborations decently. Out of the classifiers, we selected for our study, Random Forest is found to perform consistently better than other classifiers. However, except SVM and J48, the performance of other classifiers e.g. Logistic Regression, Logistic Regression Tree and Bayes Net is also close to Random

Table 6: Result for RQ1:Prediction of future collaborations among developers based on comment data

Project	Classifier	Precision	Recall	F-measure	AUROC
Hibernate	J48(Decision Tree)	0.881	0.881	0.881	0.848
	Random Forest	0.853	0.853	0.853	0.903
	Logistic Model Tree	0.853	0.853	0.853	0.901
	Logistic Regression	0.804	0.803	0.803	0.873
	SMO(Support Vector Machine)	0.854	0.853	0.852	0.853
	BayesNet	0.861	0.86	0.86	0.9
Spark	J48(Decision Tree)	0.796	0.796	0.796	0.811
	Random Forest	0.799	0.799	0.799	0.863
	Logistic Model Tree	0.8	0.8	0.8	0.851
	Logistic Regression	0.777	0.775	0.774	0.839
	SMO(Support Vector Machine)	0.785	0.783	0.783	0.783
	BayesNet	0.8	0.8	0.8	0.86
Flink	J48(Decision Tree)	0.776	0.775	0.774	0.752
	Random Forest	0.767	0.766	0.766	0.813
	Logistic Model Tree	0.787	0.777	0.775	0.821
	Logistic Regression	0.782	0.772	0.77	0.821
	SMO(Support Vector Machine)	0.799	0.783	0.78	0.783
	BayesNet	0.798	0.793	0.793	0.805
Kafka	J48(Decision Tree)	0.744	0.74	0.74	0.768
	Random Forest	0.759	0.759	0.759	0.851
	Logistic Model Tree	0.764	0.759	0.758	0.832
	Logistic Regression	0.755	0.746	0.744	0.816
	SMO(Support Vector Machine)	0.748	0.733	0.729	0.733
	BayesNet	0.756	0.755	0.755	0.835
Wildfly	J48(Decision Tree)	0.737	0.737	0.737	0.715
	Random Forest	0.752	0.752	0.752	0.829
	Logistic Model Tree	0.749	0.749	0.749	0.816
	Logistic Regression	0.755	0.749	0.748	0.814
	SMO(Support Vector Machine)	0.753	0.752	0.752	0.752
	BayesNet	0.77	0.77	0.77	0.845

Forest. Between comment based DSN and commit based DSN, the performance of the classifiers is better on comment data (approximate average AUROC value 0.82) than commit data (approximate average AUROC Value 0.70). However, the approximate average recall value of classifiers on commit based DSN (0.81) is slightly higher than comment based DSN (0.783).

Table 7: Result for RQ1: Prediction of future collaborations among developers based on commit data

Dataset	**Classifier**	**Precision**	**Recall**	**F-measure**	**AUROC**
Hibernate	J48(Decision Tree)	0.878	0.887	0.882	0.773
	Random Forest	0.869	0.898	0.877	0.847
	Logistic Model Tree	0.85	0.887	0.864	0.853
	Logistic Regression	0.902	0.909	0.905	0.683
	SMO(Support Vector Machine)	0.851	0.897	0.863	0.522
	BayesNet	0.921	0.849	0.873	0.841
Spark	J48(Decision Tree)	0.815	0.859	0.818	0.605
	Random Forest	0.834	0.866	0.828	0.757
	Logistic Model Tree	0.825	0.863	0.817	0.686
	Logistic Regression	0.827	0.864	0.813	0.691
	SMO(Support Vector Machine)	0.739	0.738	0.736	0.667
	BayesNet	0.81	0.75	0.774	0.677
Flink	J48(Decision Tree)	0.768	0.8	0.766	0.68
	Random Forest	0.762	0.792	0.767	0.765
	Logistic Model Tree	0.761	0.796	0.757	0.689
	Logistic Regression	0.777	0.805	0.769	0.662
	SMO(Support Vector Machine)	0.781	0.796	0.722	0.529
	BayesNet	0.775	0.78	0.778	0.687
Kafka	J48(Decision Tree)	0.737	0.761	0.729	0.618
	Random Forest	0.752	0.771	0.746	0.722
	Logistic Model Tree	0.714	0.747	0.702	0.682
	Logistic Regression	0.695	0.738	0.68	0.676
	SMO(Support Vector Machine)	0.644	0.731	0.622	0.502
	BayesNet	0.711	0.702	0.706	0.679
Wildfly	J48(Decision Tree)	0.83	0.837	0.831	0.776
	Random Forest	0.853	0.859	0.852	0.874
	Logistic Model Tree	0.83	0.838	0.831	0.818
	Logistic Regression	0.757	0.78	0.731	0.735
	SMO(Support Vector Machine)	0.751	0.766	0.680	0.522
	BayesNet	0.809	0.792	0.799	0.818

RQ2: How useful are the recommended developer-developer links in terms of improving the social capital of the developers? Since we are investigating the usefulness of the link prediction scheme on improving the social capital of developers, results of our experiments towards answering this research question are shown in Table 8 and

Table 8: Recall of Gains in various measures characterizing the Social Capital (Based on Comment Data)

Project	**Classifier**	**RGCI**	**RGADC**	**RGAGI**	**RGADI**	**RGAAI**
Flink	BayesNet	0.536	0.682	0.53	0.31	0.684
	J48(Decision Tree)	0.548	0.654	0.527	0.323	0.653
	Logistic Model Tree	0.569	0.71	0.568	0.351	0.713
	Logistic Regression	0.577	0.716	0.575	0.349	0.719
	Random Forest	0.515	0.671	0.517	0.296	0.674
	SMO(Support Vector Machine)	0.603	0.737	0.604	0.376	0.74
Hibernate	BayesNet	0.507	0.603	0.574	0.451	0.595
	J48(Decision Tree)	0.521	0.593	0.576	0.481	0.582
	Logistic Model Tree	0.549	0.613	0.592	0.501	0.605
	Logistic Regression	0.521	0.617	0.598	0.483	0.612
	Random Forest	0.507	0.594	0.563	0.447	0.588
	SMO(Support Vector Machine)	0.563	0.619	0.603	0.51	0.611
Kafka	BayesNet	0.526	0.574	0.487	0.326	0.592
	J48(Decision Tree)	0.552	0.605	0.52	0.341	0.62
	Logistic Model Tree	0.552	0.605	0.515	0.352	0.622
	Logistic Regression	0.59	0.671	0.585	0.413	0.685
	Random Forest	0.488	0.536	0.449	0.293	0.557
	SMO(Support Vector Machine)	0.621	0.708	0.615	0.428	0.717
Spark	BayesNet	0.511	0.36	0.402	0.332	0.338
	J48(Decision Tree)	0.496	0.397	0.393	0.323	0.415
	Logistic Model Tree	0.491	0.391	0.38	0.313	0.41
	Logistic Regression	0.53	0.384	0.423	0.349	0.359
	Random Forest	0.5	0.451	0.419	0.325	0.466
	SMO(Support Vector Machine)	0.535	0.421	0.424	0.35	0.431
WildFly	BayesNet	0.564	0.602	0.539	0.362	0.626
	J48(Decision Tree)	0.545	0.526	0.507	0.349	0.535
	Logistic Model Tree	0.539	0.553	0.511	0.343	0.567
	Logistic Regression	0.612	0.615	0.58	0.421	0.636
	Random Forest	0.515	0.534	0.499	0.332	0.549
	SMO(Support Vector Machine)	0.576	0.597	0.542	0.376	0.621

Table 9. This is evident from the tables that the social capital of the projects can be improved significantly using the proposed method. As can be seen, the approximate average RGSC (average on all the classifiers and all the 5 concrete measures of social capital) achieved on comment data is 0.52 while on commit data we can achieve the average RGSC value of 0.82. It should also be noted that though ADC and ADI look similar in nature, the proposed method is able to improve ADC more than ADI which suggests that new links recommended by the method can improve the reachability of the

Table 9: Recall of Gains in various measures characterizing the Social Capital (Based on Commit Data)

Project	**Classifier**	**RGCI**	**RGADC**	**RGAGI**	**RGADI**	**RGAAI**
Flink	BayesNet	0.788	0.843	0.828	0.669	0.852
	J48(Decision Tree)	0.889	0.92	0.909	0.824	0.926
	Logistic Regression Tree	0.929	0.943	0.941	0.869	0.943
	Logistic Regression	0.904	0.93	0.931	0.843	0.934
	Random Forest	0.859	0.891	0.878	0.755	0.893
	SMO(Support Vector Machine)	0.722	0.824	0.737	0.687	0.824
Hibernate	BayesNet	0.667	0.781	0.792	0.649	0.808
	J48(Decision Tree)	0.944	0.937	0.938	0.95	0.935
	Logistic Regression Tree	0.778	0.844	0.861	0.798	0.861
	Logistic Regression	0.778	0.844	0.861	0.798	0.861
	Random Forest	0.778	0.844	0.861	0.798	0.861
	SMO(Support Vector Machine)	0.333	0.056	0.196	0.466	0.044
Kafka	BayesNet	0.765	0.785	0.791	0.663	0.783
	J48(Decision Tree)	0.856	0.885	0.887	0.769	0.888
	Logistic Regression Tree	0.891	0.912	0.919	0.81	0.913
	Logistic Regression	0.94	0.95	0.946	0.89	0.949
	Random Forest	0.884	0.902	0.921	0.816	0.903
	SMO(Support Vector Machine)	0.4	0.562	0.471	0.379	0.586
Spark	BayesNet	0.767	0.801	0.783	0.674	0.805
	J48(Decision Tree)	0.97	0.975	0.982	0.955	0.976
	Logistic Regression Tree	0.978	0.979	0.983	0.969	0.98
	Logistic Regression	0.986	0.988	0.99	0.979	0.988
	Random Forest	0.98	0.98	0.984	0.973	0.98
	SMO(Support Vector Machine)	0.734	0.818	0.743	0.693	0.792
WildFly	BayesNet	0.711	0.718	0.96	0.416	0.698
	J48(Decision Tree)	0.801	0.806	0.973	0.555	0.792
	Logistic Regression Tree	0.809	0.813	0.981	0.567	0.8
	Logistic Regression	0.923	0.925	0.962	0.804	0.919
	Random Forest	0.812	0.816	0.943	0.582	0.804
	SMO(Support Vector Machine)	0.702	0.781	0.963	0.699	0.783

developer by connecting him/her to a new set of peer developers. Good value of RGAGI indicates that new links predicted by our method are also influential and important as by adding them in the network, the node may connect unconnected pair or set of the developers. Good value of RGCI and RGAAI indicate that the proposed method can not only improve the direct connections (path of length 1) among developers but also can reduce the length of the shortest paths among them. In nutshell, the proposed method has the potential to improve the social capital of the project from many perspectives.

RQ3: How does the performance of link prediction vary across different types of DSNs in improving the social capital i.e. comment data based DSN or commit data based DSN ? What explains this difference in performance?

Answer to this research question is riveting. By comparing the Table 6 and Table 7, it can be seen that the performance of classifiers are better on comment based DSN. However, the RGSC value is far more in commit based DSN than comment data. This suggests that higher accuracy in link prediction does not always imply that these links are useful in improving social capital. That is, though we achieve better accuracy in predicting new association at comment level, the links predicted at the commit level are more useful in improving the social capital of the project. One possible reason for this could be that collaborations at VCS level are more technical in nature than at ITS level. Two developers who commit on the same source file/package are more likely to have common expertise (and hence would like to have a professional relationship) than those who comment on the same issue report. Comments can also be made by software users which may otherwise not be interested in forming a professional relationship with each other. To investigate this further, we also compared commit based DSN and comment

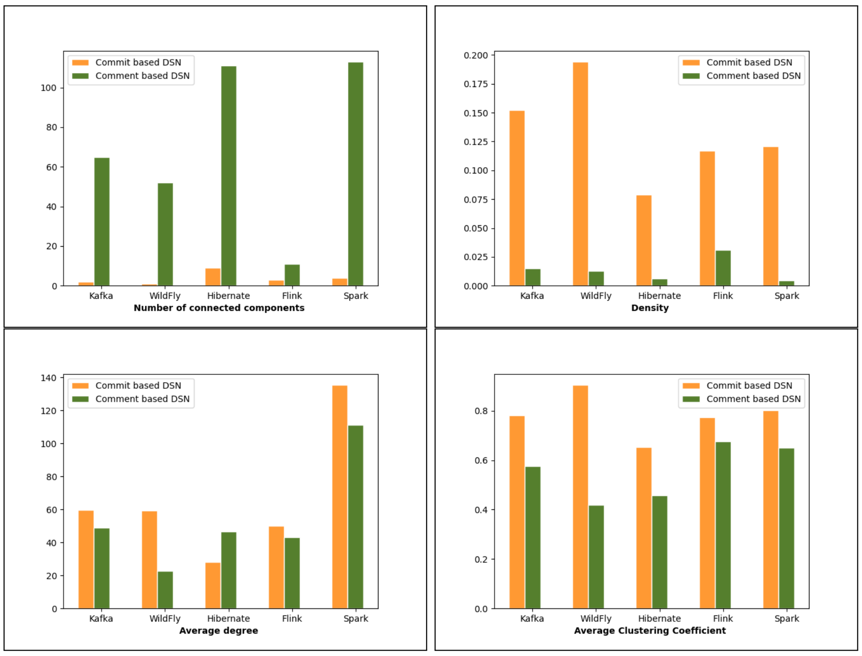

Figure 8: Comparison of Network Characteristics of Commit based-DSN and Comment based-DSN

based DSN based on network metrics described in section 4.2. (i.e. Number of connected components (NCC), Average Clustering Coefficient (ACC), Average Degree and Density of the graph). It is evident from Figure 8 that commit based DSN is more connected and cohesive than comment based DSN. The results presented in Figure 8, Table 8 and Table 9 suggest that though it is harder to predict new collaborations accurately if developers are cohesively and densely connected to each other, yet the predicted links/collaborations are more useful to recall the improvement in the social capital of the project.

In nutshell, the proposed approach can predict future collaborations among developers with decent accuracy. If these predicted links are recommended to developers, the social capital of developers and the project could increase significantly.

6 Implementation detail and Performance

In this section, we discuss the implementation detail and scalability of our approach. In particular, we discuss how we optimized the implementation to make the approach more tractable and scalable.

6.1 Theoretical Analysis

In this section, we discuss the theoretical time and space complexity of our approach. Since our study is around answering three research questions, we present our analysis based on these research questions. For the first research question(RQ-1), our entire approach can be decomposed into three phases- graph construction, computing the feature set for classifiers and running the classifiers on the training and testing data composed of this feature set.

Homogeneous graph construction using adjacency list implementation takes $O(E)$ time where E is the number of edges in the graph. In adjacency list implementation of the graph, a graph is denoted as an array of linked lists. This applies to both commits based Homo-DSN as well as comment based Homo-DSN. From the basic understanding of the graph theory, we know that the space complexity of adjacency list implementation of the homogeneous graph is $O(E + V)$ where V and E are the numbers of nodes and edges in the graph respectively.

To implement Hetero-DSN, we maintain a set of adjacency lists. This set of adjacency lists is determined by the set of meta paths that have been used in our approach. All of the meta paths used in our work (defined in table 3.3.2 and table 3.3.2) are symmetric. Symmetry and concatenability of meta paths are two properties that are very helpful in optimizing the run time complexity of our approach. A symmetric meta path represents a symmetric relationship between end objects. For instance, D-I-C-I-D of table 3.3.2 represents a symmetric relationship i.e. two developers commenting on issues found in the same component. Two meta paths, $A_1 - A_2 - A_3 - ....A_n$ and $B_1 - B_2 - B_3 - ....B_m$ are concatenable if A_n=B_1. In our work, symmetric paths are of two categories i.e. odd length meta paths (where the number of node types in the meta path is odd in number) and even length meta paths (where the number of node types in meta path is even in number). The odd length meta paths are of the form $A_1 - A_2 - A_3........A_{n-1} - A_n - A_{n-1}........A_3 - A_2 - A_1$ while the even length meta paths are of the form $A_1 - A_2 - A_3........A_n - B - B - A_n........A_3 - A_2 - A_1$. For each odd length meta path like $A_1 - A_2 - A_3........A_{n-1} - A_n - A_{n-1}........A_3 - A_2 - A_1$, we build $n-1$ adjacency lists capturing the relationship between A_i and A_{i+1}. Here, 1≤i≤$n-1$. Similarly, for each even length meta path like $A_1 - A_2 - A_3........A_n - B - B - A_n........A_3 - A_2 - A_1$, we build n adjacency lists. Out of these n adjacency lists, $n - 1$ lists capture the relationship between A_i and A_{i+1} where 1≤i≤$n - 1$ and one list is between A_n and B. In general, we can say that we might have to create $O(n)$ adjacency lists for the types of meta

paths that have been considered. If the number of meta paths is m then in total we may have to create $O(m \times n)$ adjacency lists in the worst case. It should be noted that each adjacency list captures a different semantic relationship between two types of nodes. For instance, corresponding to the meta path D-I-C-I-D of table 3.3.2, we maintain two adjacency lists, one between D and I and another between I and C. The adjacency list between D-I captures the information about issues, each developer has commented upon. The adjacency list I-C records the components which the issues spanned over (the issue spans over at least one component. However, there can be a multi-component issue as well). Similarly, corresponding to a meta path D-CIT-I-I-CIT-D of table 3.3.2, we require two adjacency lists i.e. D-CIT and CIT-I. The adjacency list D-CIT records the set of commits made by the developers while the adjacency list CIT-I records the issue which has been fixed using the contributions made through the commits. If we denote the number of vertices and edges in i^{th} adjacency list by V_i and E_i respectively and $V_{max} = max(V_1, V_2, V_3....V_{mn})$ and $E_{max} = max(E_1, E_2, E_3....E_{mn})$ then total time complexity of creating the heterogeneous DSN is $O(m \times n \times E_{max})$.
Similarly the space complexity in worst case is $O(m \times n \times (V_{max} + E_{max}))$.

The second phase of answering the research question-1 (RQ-1) requires the computation of many features which are then used to train and test the classifiers. The first kind of features is computed from homogeneous graphs (Homo-DSNs based on commit data and comment data). The common neighbour count for all pairs takes $O(Vz^3)$ time [Martínez et al., 2016], where V is a number of nodes and z is the maximum degree of a node in the graph. Since Adamic-Adar and Jaccard Coefficients are based on common neighbors count, their time complexity is also $O(Vz^3)$. However, since Preferential Attachment requires less computation, it can be performed in $O(Vz^2)$. Since the length of the shortest path between all pairs of nodes is computed using Floyd-Warshall's algorithm, it can be computed in $O(V^3)$. Therefore total time complexity of this set of features is $max(O(Vz^3),O(Vz^2),O(V^3))$, which turns out to be $O(V^3)$. Since all the features except the length of the shortest path can be computed without consuming additional space and the Floyd-Warshall algorithm consumes $O(V^2)$ space, the space complexity remains in $O(V^2)$
The next set of features are defined for the heterogeneous developers' social network. It should be noted again that there can exist many real concrete paths between two developer nodes following a specific meta path. In general, we denote a type of object (node) by a capital letter and a concrete object (node) by a lowercase letter. For instance, a concrete path $d_1 i_1 c i_2 d_2$, following a meta path DICID may exist between d_1 and d_2 if they comment on two different issues (i_1,i_2) which are found in the same component c. $|X|$ denotes the number of concrete objects (nodes) of node type X. The worst-case complexity of computing the path counts between two objects by a brute force method will be either $O(|A_n| \times \prod_{i=1}^{n-1} |A_i|^2)$ (if the meta path is of the form $A_1 - A_2 - A_3........A_{n-1} - A_n - A_{n-1}........A_3 - A_2 - A_1$) or $O(|B|^2 \times \prod_{i=1}^{n} |A_i|^2)$ (if the meta path is of the form $A_1 - A_2 - A_3........A_n - B - B - A_n........A_3 - A_2 - A_1$). This is very expensive. However, by utilizing two properties of the meta paths i.e. symmetry and concatenability, we can reduce this time complexity quite significantly. We now first discuss a few definitions and notations, which we use to illustrate how we improved the computational performance of our proposed approach. After that, we analyse the time complexity of our alternative method of computing the path counts. For space complexity, apart from maintaining $O(m \times n)$ adjacency lists as mentioned earlier, we may require one path count adjacency list which may require $O(V_{max} + E_{max})$ space in the worst case.

1. **Type**(x)**:** This returns the type of node for concrete node x. For example, Type(d) returns D, if d is a specific developer node and D denotes the type of all developer nodes.

2. **Path count**$(x, y|\rho)$**:** This denotes the number of paths existing between concrete nodes x and y by following a path ρ.

3. **Path count adjacency list**$(X, Y|\rho)$**:** This denotes the array of linked lists where each linked list records the number of path counts, from a certain concrete node e.g. x of type X to all the concrete nodes of type Y, which are connected to node x by following a meta path ρ. This way, the path count information (based on meta path ρ) among all nodes of type X and Y is recorded in this Path count adjacency list.

4. **Path count**$(x, y|XY)$ =1 if y is present in the linked list of x. if y is not present in the linked list of x then value returned by the function is 0. Here, Type(x)=X and Type(y)=Y.

5. **Path count**$(x, z|X\rho\text{YZ}) = \sum\limits_{\substack{\forall y_i: \\ Type(y_i)=Y}}$ Path count$(x, y_i|\rho) \times$ Path count$(y_i z|YZ)$.

We use the concatenability property of meta paths in the above equation (5). We compute the path counts between two nodes by breaking the composite meta paths. That is, we can use the above equation to compute the path counts between node x of type X and node z of type Z (by following a meta path $X\rho$YZ). The above equation (5) is key to reduce the time complexity of computing path counts between two nodes.
To compare this method with the brute force method, lets assume that we have to compute all pair path counts for nodes of type A_1 using the two types of the symmetric meta paths i.e. $A_1 - A_2 - A_3........A_{n-1} - A_n - A_{n-1}.......A_3 - A_2 - A_1$ and $A_1 - A_2 - A_3........A_n - B - B - A_n........A_3 - A_2 - A_1$. In both the types of meta paths, we can use definition 4 and equation 5 successively to produce the Path count adjacency list defined in definition 3. For instance, to compute the Path count adjacency list between A_1 and A_1 using $A_1 - A_2 - A_3........A_{n-1} - A_n - A_{n-1}.......A_3 - A_2 - A_1$ meta path, we can first compute Path count adjacency list between A_1 and A_3 using meta path $A_1A_2A_3$. This takes $O(|A_1| \times |A_2| \times |A_3)$ time. Then we use this Path count adjacency list and adjacency list between A_3 and A_4 to compute the Path count adjacency list between A_1 and A_4. This takes $O(|A1| \times |A_3| \times |A_4|)$ time. This way, to compute the Path count adjacency list between A_1 and A_n takes $O(\sum\limits_{i=2}^{n-1} |A_1| \times |A_i| \times |A_{i+1}|)$. However, to compute the path counts between two concrete objects of type A_1 (using $A_1 - A_2 - A_3........A_{n-1} - A_n - A_{n-1}.......A_3 - A_2 - A_1$ meta path), we might have to look up $O(|A_n|)$ entries in Path count adjacency list between A_1 and A_n in worst case. Therefore, we can compute the path counts between two objects of type A_1 using meta path $A_1 - A_2 - A_3........A_{n-1} - A_n - A_{n-1}.......A_3 - A_2 - A_1$ in $O(|A_n| \times \sum\limits_{i=2}^{n-1} |A1| \times |A_i| \times |A_{i+1}|)$ time. Similarly the time complexity due to second type of meta path $(A_1 - A_2 - A_3........A_n - B - B - A_n........A_3 - A_2 - A_1)$ could be proved as $O(|B| \times ((\sum\limits_{i=2}^{n-1} |A1| \times |A_i| \times |A_{i+1}|) + |A1| \times |A_n| \times |B|))$. For both types of meta paths, it is a significant improvement over the brute force method. Apart from keeping the required adajacency lists, we require to keep one path count adjacency lists

for our calculations and hence the space complexity is $O(m \times n \times (V_{max} + E_{max}))$ $+O(V_{max} + E_{max})$ $=O(m \times n \times (V_{max} + E_{max}))$.
Computing contribution-based features (section 3.3.3) is relatively easy. For all the features from 3.3.3.1 to 3.3.3.3, we simply require a single scan of issue reports, issue comment data and commit data and build the dictionaries at the same time. For the dictionaries, the developers are the keys and total contributions are the values. For instance, the dev_issue dictionary maintains developers as key and the number of issue reports which he has commented/committed for, are the values. Fetching a value from the dictionary for a certain developer takes $O(1)$ time while building such dictionaries take $O(|comments| + |issues| + |commits|)$ time. Here, $|comments|$,$|issues|$ and $|commits|$ are total number of comments , total number of issues and total number of commits present in our dataset. Apart from the dictionaries (of size $O(V)$ each), we do not require any space and hence only $O(V)$ is added to the space complexity. For feature 3.3.3.4, we can use the homogeneous networks and hence the number of elements in the linked list of a certain developer in the adjacency list of the graph can return the required information. No extra space complexity is required for this. For TF-IDF based features, we require to compute the TF-IDF matrix. The details of the time and space complexity could be found in [Manning et al., 2008].

In the third phase of computation for RQ-1, we used Weka [Hall et al., 2009], a popular machine learning and data mining toolkit. We used popular machine learning classifiers in our study. A detailed study about their time and space complexity is available in literature [Hassine et al., 2019] [Cooper, 1990] [Su and Zhang, 2006] [Landwehr et al., 2005] [Bulso et al., 2019] [Abdiansah and Wardoyo, 2015] [Sani et al., 2018].

For the second research question (RQ-2), apart from the computations done for RQ-1, we require to compute various measures of social capital. These measures are network-based measures and are to be computed on homogeneous graphs of training and test periods. Cohesion Index takes $O(V^3)$ time if the number of vertices is V in the graph. This is because we have to investigate $O(V^2)$ pairs of developers if they are adjacent to each other or not. However, to determine this adjacency between developers we may have to traverse the entire linked list of the developer i.e. requiring $O(V)$ time in the worst case. For Average Decay Centrality, with little modification, we can use BFS to find the k-reachable nodes (nodes that are reachable using k hops) from the certain node. Since BFS takes $O(V + E)$ time, total complexity turns out to be $O(V \times m \times (V + E))$. Average Godfather Index takes $O(V^3)$ time as investigating the pairwise connectivity between nodes takes $O(V^2)$ time (inner summation) and this has to be investigated for each node (outer summation). Average Diffusion Index is the costliest operation because it involves matrix multiplication. To compute q^{th} power of the matrix takes $O(q \times V^3)$ time. Since all entries of it are to be summed up, the total time complexity of it is $O(m \times V^5)$. The shortest distance between two nodes of an undirected graph can be computed in $O(V + E)$ time. However, we may have to compute this $O(V^2)$ times for the calculation of Average Agility Index and hence the time complexity of AAI is $O(V^2(V + E))$. Since no extra space (other than the adjacency list of the homogeneous graph) is required, the additional space complexity to compute all measures of social capital remains in $O(1)$

For RQ-3, we require to compute some network properties. For the degree of the node, in the worst case, the number of neighbours of a particular node can be computed in $O(V)$ time. Since average is to be taken for all the nodes for computing the average degree, it takes $O(V^2)$ time in total. Similarly, it is easy to see that number of connected components in a graph can be found in $O(V + E)$ time using the DFS traversal algorithm.

For density, we require to traverse the graph to find the number of edges that takes $O(E)$ time. Hence the time complexity of the Average degree computation takes only $O(E)$ time. For average clustering coefficient as defined in [Kaiser, 2008], we first have to compute the number of edges between the neighbours of each node. Since in the worst case, the number of neighbours of a node could be V. To find edges between them can take $O(V^2)$ time. The degree of a node can be computed in $O(V)$ time. Hence for individual nodes, computational effort of $O(V^2)$ may be required. Since we have to make this effort for each node, the worst-case time to compute average clustering coefficient is $O(V^3)$. Since all computations are to be performed on the homogeneous graph, we do not require any additional space and hence space complexity remains to be in $O(1)$.

6.2 Pragmatic Analysis

The above theoretical analysis seems to suggest that the computational task involved in the study is practically intractable. However, in practice, we could perform the computations more efficiently. One of the reasons is that the real graph constructed out of data is sparse while the worst-case analysis done in the previous section assumed the graph as quite dense. For example, many times, we considered the number of neighbors of the node as V (the set of nodes in the graph) while in reality, the number of neighbors of a node is far less than the total number of nodes. Second, in the case of a heterogeneous graph, we took the general case for the meta paths where the length of meta paths was considered arbitrarily large. In reality, we found that meta paths longer than 9 are not able to improve the performance significantly, and hence the maximum length of our meta path is only 9 (a constant). Similarly, the meta paths are chosen before computing the path count measures and hence number of meta paths is also constant. In our case, the total number of meta paths is 12. It should also be noted that since parts of the meta paths are overlapping with each other, we do not require to maintain the adjacency lists exclusively and separately for each meta path. For instance, the adjacency list D-I for comment data (table 3.3.2) and D-CIT (table 3.3.2) for commit data are part of many meta paths, and hence only a single copy of such adjacency list may be created and maintained. This way, time and space complexity could be reduced significantly. Since the number of nodes in the homogeneous network i.e. developers in the project are not beyond a few hundred and the graph is sparse enough, the social capital measures also could be computed in a reasonable time. For our experiments, we used a machine with 16 GB RAM, an Intel-i5 processor with a clock frequency of 2.10 GHz, and with 512 GB SSD drive. Running the entire sequence of tasks (for all the research questions) takes 2 hours (WildFly) to 21 hours (Spark) depending on the size of project data.

7 Threats to Validity

Though we have conducted our experiments on 5 popular and diverse OSS projects to make our study generalizable, we could not perform our study on all 33 projects of the SEOSS dataset. This leaves the study, limited to only a few projects and hence puts a threat to its validity. The prime reason for restricting our study to only 5 projects is that developers have multiple identities in different types of datasets. That is, many times, the same developer uses separate identities to version control systems and bug tracking systems. Since our study is around developer profiling (helping them to connect with each other for a task and measuring and improving their social capital), it is very essential

to be accurate in their identification. Matching the identities of developers from both types of datasets i.e. VCS and ITS is not trivial. For instance, there are more than 40000 comments and more than 7000 commits in Spark's ITS and VCS data respectively. Each comment and commit has the developer associated with it. Therefore, we require to match 40000×7000 entries in total. Fortunately, some entries match exactly and hence can be filtered out from manual search space. However, the unmatched and ambiguous entries are significant in number (more than 10 %) in the data set. That is even after performing auto filtration, more than 4000×700 matches may require removing ambiguity from the data. This requires significant manual effort and hence we restricted our study to only 5 projects. Automated identity merging is an active research area and many automated tools have been developed to solve this problem. However, these tools are not perfect and we can not rely on them.
In our study, we consider every OSS contributor as a developer. In fact, developers and contributors are used synonymously in our study. We made this assumption particularly because work division among OSS contributors in Open Source projects is not as strict as in closed source projects and hence many times, OSS contributors play multiple roles. Though many past studies in the software engineering field have also made this assumption [Hong et al., 2011] [Zhang et al., 2013], in practice, especially in a closed source environment, this assumption may not be true. Many times, the roles/positions of contributors are not specified on the ITS and VCS platforms and hence it also creates a hindrance for the researchers to perform the role-based study. For instance, comments can be made by users of the software, developer, quality analyst, reporter, integrator, etc. Similarly, the commits can be made by an integrator or tester. However, the roles and organizational positions of the commenters and the committers are not specified in the ITS and VCS dataset. Though our study is towards helping OSS contributors to be connected with each other and helping them to improve their social capital, ignoring the roles of these contributors puts a threat to the validity of the study.

8 Conclusion

Social capital of the project is instrumental in attracting the sustained participating of the developers. In this paper, an approach to improve the developers individual social capital has been proposed and the cumulative effect of this improved individual social capital to elevate the social capital of the project is also established. The proposed approach uses three DSNs -commit based Homo-DSN, comment based Homo-DSN and integrated Hetero-DSN to model the collaborations among developers on ITS, and VCS platforms. The proposed approach uses network proximity and developer activity based features to build the classifier for predicting the future collaborations among developers. Such predicted collaborations can be recommended to the developers which can help them to improve their social ties and social capital. Proposed approach established that the predicted links are significantly useful in improving the social capital at the project level. Through experimental analysis, it is evident that the prediction accuracy of predicting future collaborations is better on ITS data than VCS data. Likewise, new links predicted for VCS data are more useful in improving the social capital of developers. Results of the experiments also establish that accuracy of predicted links and their importance are correlated with the values of global network properties capturing the network structure of DSN.

ACKNOWLEDGMENT

First author thanks Mahen Gandhi (Swiggy Labs, Bangalore) for his valuable inputs in validating the relevance of the problem in its early stage.

References

[Abdiansah and Wardoyo, 2015] Abdiansah, A. and Wardoyo, R. (2015). Time complexity analysis of support vector machines (svm) in libsvm. *International journal computer and application*, 128(3):28–34.

[Abreu and Premraj, 2009] Abreu, R. and Premraj, R. (2009). How developer communication frequency relates to bug introducing changes. In *Proceedings of the joint international and annual ERCIM workshops on Principles of software evolution (IWPSE) and software evolution (Evol) workshops*, pages 153–158.

[Aguilera, 2002] Aguilera, M. B. (2002). The impact of social capital on labor force participation: Evidence from the 2000 social capital benchmark survey. *Social science quarterly*, 83(3):853–874.

[Akula et al., 2019] Akula, R., Yousefi, N., and Garibay, I. (2019). Deepfork: Supervised prediction of information diffusion in github. *arXiv preprint arXiv:1910.07999*.

[Al Hasan et al., 2006] Al Hasan, M., Chaoji, V., Salem, S., and Zaki, M. (2006). Link prediction using supervised learning. In *SDM06: workshop on link analysis, counter-terrorism and security*, volume 30, pages 798–805.

[Al Hasan and Zaki, 2011] Al Hasan, M. and Zaki, M. J. (2011). A survey of link prediction in social networks. In *Social network data analytics*, pages 243–275. Springer.

[Amreen et al., 2020] Amreen, S., Mockus, A., Zaretzki, R., Bogart, C., and Zhang, Y. (2020). Alfaa: Active learning fingerprint based anti-aliasing for correcting developer identity errors in version control systems. *Empirical Software Engineering*, pages 1–32.

[Banitaan and Alenezi, 2013] Banitaan, S. and Alenezi, M. (2013). Decoba: Utilizing developers communities in bug assignment. In *2013 12th International Conference on Machine Learning and Applications*, volume 2, pages 66–71. IEEE.

[Bhattacharya et al., 2012] Bhattacharya, P., Iliofotou, M., Neamtiu, I., and Faloutsos, M. (2012). Graph-based analysis and prediction for software evolution. In *2012 34th International Conference on Software Engineering (ICSE)*, pages 419–429. IEEE.

[Bhattacharya and Neamtiu, 2010] Bhattacharya, P. and Neamtiu, I. (2010). Fine-grained incremental learning and multi-feature tossing graphs to improve bug triaging. In *2010 IEEE International Conference on Software Maintenance*, pages 1–10. IEEE.

[Bird et al., 2006] Bird, C., Gourley, A., Devanbu, P., Gertz, M., and Swaminathan, A. (2006). Mining email social networks. In *Proceedings of the 2006 international workshop on Mining software repositories*, pages 137–143.

[Bird et al., 2009] Bird, C., Nagappan, N., Gall, H., Murphy, B., and Devanbu, P. (2009). Putting it all together: Using socio-technical networks to predict failures. In *2009 20th International Symposium on Software Reliability Engineering*, pages 109–119. IEEE.

[Brown and Ferris, 2007] Brown, E. and Ferris, J. M. (2007). Social capital and philanthropy: An analysis of the impact of social capital on individual giving and volunteering. *Nonprofit and voluntary sector quarterly*, 36(1):85–99.

[Bulso et al., 2019] Bulso, N., Marsili, M., and Roudi, Y. (2019). On the complexity of logistic regression models. *Neural computation*, 31(8):1592–1623.

[Burt, 1998] Burt, R. S. (1998). The gender of social capital. *Rationality and society*, 10(1):5–46.

[Canfora et al., 2012] Canfora, G., Di Penta, M., Oliveto, R., and Panichella, S. (2012). Who is going to mentor newcomers in open source projects? In *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*, pages 1–11.

[Casalnuovo et al., 2015] Casalnuovo, C., Vasilescu, B., Devanbu, P., and Filkov, V. (2015). Developer onboarding in github: the role of prior social links and language experience. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, pages 817–828.

[Chiu et al., 2006] Chiu, C.-M., Hsu, M.-H., and Wang, E. T. (2006). Understanding knowledge sharing in virtual communities: An integration of social capital and social cognitive theories. *Decision support systems*, 42(3):1872–1888.

[Chou and He, 2011] Chou, S.-W. and He, M.-Y. (2011). The factors that affect the performance of open source software development–the perspective of social capital and expertise integration. *Information Systems Journal*, 21(2):195–219.

[Cooper, 1990] Cooper, G. F. (1990). The computational complexity of probabilistic inference using bayesian belief networks. *Artificial intelligence*, 42(2-3):393–405.

[Daniel et al., 2018] Daniel, S., Midha, V., Bhattacherhjee, A., and Singh, S. P. (2018). Sourcing knowledge in open source software projects: The impacts of internal and external social capital on project success. *The Journal of Strategic Information Systems*, 27(3):237–256.

[Datta et al., 2011] Datta, S., Sindhgatta, R., and Sengupta, B. (2011). Evolution of developer collaboration on the jazz platform: a study of a large scale agile project. In *Proceedings of the 4th India Software Engineering Conference*, pages 21–30.

[Diaz-Pace et al., 2018] Diaz-Pace, J. A., Tommasel, A., and Godoy, D. (2018). Can network analysis techniques help to predict design dependencies? an initial study. In *2018 IEEE International Conference on Software Architecture Companion (ICSA-C)*, pages 64–67. IEEE.

[Díaz-Pace et al., 2018] Díaz-Pace, J. A., Tommasel, A., and Godoy, D. (2018). Towards anticipation of architectural smells using link prediction techniques. In *2018 IEEE 18th International Working Conference on Source Code Analysis and Manipulation (SCAM)*, pages 62–71. IEEE.

[Guiso et al., 2004] Guiso, L., Sapienza, P., and Zingales, L. (2004). The role of social capital in financial development. *American economic review*, 94(3):526–556.

[Hahn et al., 2008] Hahn, J., Moon, J. Y., and Zhang, C. (2008). Emergence of new project teams from open source software developer networks: Impact of prior collaboration ties. *Information Systems Research*, 19(3):369–391.

[Hall et al., 2009] Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., and Witten, I. H. (2009). The weka data mining software: an update. *ACM SIGKDD explorations newsletter*, 11(1):10–18.

[Hassine et al., 2019] Hassine, K., Erbad, A., and Hamila, R. (2019). Important complexity reduction of random forest in multi-classification problem. In *2019 15th International Wireless Communications & Mobile Computing Conference (IWCMC)*, pages 226–231. IEEE.

[Hong et al., 2011] Hong, Q., Kim, S., Cheung, S. C., and Bird, C. (2011). Understanding a developer social network and its evolution. In *2011 27th IEEE international conference on software maintenance (ICSM)*, pages 323–332. IEEE.

[Huang et al., 2013] Huang, K., Fan, Y., Tan, W., and Li, X. (2013). Service recommendation in an evolving ecosystem: A link prediction approach. In *2013 IEEE 20th International Conference on Web Services*, pages 507–514. IEEE.

[Jackson, 2019] Jackson, M. O. (2019). A typology of social capital and associated network measures. *Social Choice and Welfare*, pages 1–26.

[Jeong et al., 2009] Jeong, G., Kim, S., and Zimmermann, T. (2009). Improving bug triage with bug tossing graphs. In *Proceedings of the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*, pages 111–120.

[Kaiser, 2008] Kaiser, M. (2008). Mean clustering coefficients: the role of isolated nodes and leafs on clustering measures for small-world networks. *New Journal of Physics*, 10(8):083042.

[Kerzazi and El Asri, 2016] Kerzazi, N. and El Asri, I. (2016). Who can help to review this piece of code? In *Working Conference on Virtual Enterprises*, pages 289–301. Springer.

[Kumar and Gupta, 2013] Kumar, A. and Gupta, A. (2013). Evolution of developer social network and its impact on bug fixing process. In *Proceedings of the 6th India Software Engineering Conference*, pages 63–72. ACM.

[Landwehr et al., 2005] Landwehr, N., Hall, M., and Frank, E. (2005). Logistic model trees. *Machine learning*, 59(1-2):161–205.

[Liben-Nowell and Kleinberg, 2007] Liben-Nowell, D. and Kleinberg, J. (2007). The link-prediction problem for social networks. *Journal of the American society for information science and technology*, 58(7):1019–1031.

[Manning et al., 2008] Manning, C. D., Schütze, H., and Raghavan, P. (2008). *Introduction to information retrieval*. Cambridge university press.

[Martínez et al., 2016] Martínez, V., Berzal, F., and Cubero, J.-C. (2016). A survey of link prediction in complex networks. *ACM computing surveys (CSUR)*, 49(4):1–33.

[Mendez et al., 2018] Mendez, C., Padala, H. S., Steine-Hanson, Z., Hilderbrand, C., Horvath, A., Hill, C., Simpson, L., Patil, N., Sarma, A., and Burnett, M. (2018). Open source barriers to entry, revisited: A sociotechnical perspective. In *Proceedings of the 40th International Conference on Software Engineering*, pages 1004–1015.

[Méndez-Durón and García, 2009] Méndez-Durón, R. and García, C. E. (2009). Returns from social capital in open source software networks. *Journal of Evolutionary Economics*, 19(2):277–295.

[Meneely et al., 2008] Meneely, A., Williams, L., Snipes, W., and Osborne, J. (2008). Predicting failures with developer networks and social network analysis. In *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering*, pages 13–23. ACM.

[Nielek et al., 2016] Nielek, R., Jarczyk, O., Pawlak, K., Bukowski, L., Bartusiak, R., and Wierzbicki, A. (2016). Choose a job you love: predicting choices of github developers. In *2016 IEEE/WIC/ACM International Conference on Web Intelligence (WI)*, pages 200–207. IEEE.

[Posnett et al., 2013] Posnett, D., D'Souza, R., Devanbu, P., and Filkov, V. (2013). Dual ecological measures of focus in software development. In *2013 35th International Conference on Software Engineering (ICSE)*, pages 452–461. IEEE.

[Qiu et al., 2019] Qiu, H. S., Nolte, A., Brown, A., Serebrenik, A., and Vasilescu, B. (2019). Going farther together: The impact of social capital on sustained participation in open source. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, pages 688–699. IEEE.

[Rath and Mäder, 2019] Rath, M. and Mäder, P. (2019). The seoss 33 dataset—requirements, bug reports, code history, and trace links for entire projects. *Data in brief*, 25:104005.

[Samad et al., 2020] Samad, A., Qadir, M., Nawaz, I., Islam, M. A., and Aleem, M. (2020). A comprehensive survey of link prediction techniques for social network. *EAI Endorsed Trans. Indust. Netw. & Intellig. Syst.*, 7(23):e3.

[Sani et al., 2018] Sani, H. M., Lei, C., and Neagu, D. (2018). Computational complexity analysis of decision tree algorithms. In *International Conference on Innovative Techniques and Applications of Artificial Intelligence*, pages 191–197. Springer.

[Singh et al., 2011] Singh, P. V., Tan, Y., and Mookerjee, V. (2011). Network effects: The influence of structural capital on open source project success. *Mis Quarterly*, pages 813–829.

[Su and Zhang, 2006] Su, J. and Zhang, H. (2006). A fast decision tree learning algorithm. In *Aaai*, volume 6, pages 500–505.

[Sun et al., 2011] Sun, Y., Barber, R., Gupta, M., Aggarwal, C. C., and Han, J. (2011). Co-author relationship prediction in heterogeneous bibliographic networks. In *2011 International Conference on Advances in Social Networks Analysis and Mining*, pages 121–128. IEEE.

[Tan et al., 2007] Tan, Y., Mookerjee, V., and Singh, P. (2007). Social capital, structural holes and team composition: Collaborative networks of the open source software community. *ICIS 2007 Proceedings*, page 155.

[Wu et al., 2011] Wu, W., Zhang, W., Yang, Y., and Wang, Q. (2011). Drex: Developer recommendation with k-nearest-neighbor search and expertise ranking. In *2011 18th Asia-Pacific Software Engineering Conference*, pages 389–396. IEEE.

[Zhang et al., 2016] Zhang, W., Wang, S., and Wang, Q. (2016). Ksap: An approach to bug report assignment using knn search and heterogeneous proximity. *Information and Software Technology*, 70:68–84.

[Zhang et al., 2013] Zhang, W., Wang, S., Yang, Y., and Wang, Q. (2013). Heterogeneous network analysis of developer contribution in bug repositories. In *2013 International Conference on Cloud and Service Computing*, pages 98–105. IEEE.