Software Developer Social Networks Analysis: A Mining Software Repository Approach

Long Ma

This report is submitted as partial fulfilment of the requirements for the Honours Programme of the School of Computer Science and Software Engineering, The University of Western Australia, 2012

Abstract

Modern software are pieces of complex work delivered collaboratively by groups of software developers. These developers and their relationships formed by co-editing the same source code file constituting a developer social network. Certainly, different projects have different shapes of developer social network which can be reflected by their statistical properties. An important observation is that the structure of the developer social network would influence the efficiency and quality of the development process because of different patterns and efficiency of the communication among developers. Therefore, it is necessary to deepen our understanding of developer social networks in the bigger social network analysis context.

In this dissertation, we address the challenge of identifying important properties that can characterize a developer social network and quantifying their influences on the project. We differentiate "healthy" networks from "problematic" networks according to whether it helps increasing development efficiency and quality. Machine learning algorithms are employed to build classification models to predict the current status of an ongoing project from developer social network aspect. These models are evaluated and compared on real world datasets. The prediction results are helpful for project managers to detect hidden risks in the current developer social networks and in the software module division.

A more important and interesting question is: given the current status of a developer social network, how would we optimize it in order to increase its "healthiness". In this research, we use recommendation algorithms to recommend collaborative relationships to the developers with the aim of optimizing the network. We tried random walk and ant colony recommendation algorithms which are fine-tuned for our task.

Finally, we implement a system that integrates developer social network construction, visualization, analysis and recommendation functionalities proposed in this dissertation. This system helps researchers and project managers to identify existing problems with current developer social networks and provide suggestions by recommending appropriate relationships between developers to optimize the network. The system is implemented using Java and other third party utilities.

Developer social network is a particular type of social network, which has its own properties such as its dynamics and its bipartite-graph nature. Our research explores some previously unexplored topics which will certainly deepen our understanding of developer social networks and provide a useful tool for developer social network analysis and optimization.

Keywords: MSR, Social network, Data mining, Classification, Recommendation, Software engineering
CR Categories: H.2.8, H.3.3

Acknowledgements

First of all, I would like to express my appreciation to my supervisor, Prof. Amitava Datta for his patience and guidance. During the whole process of my dissertation, you were always there giving me valuable suggestions and expert knowledge. Your research enthusiasm hugely encouraged me. Thanks for acting as a friend to discuss the project with me. I really appreciate to have an supervisor like you. Thanks for all the help.

My gratitude also goes to Professor Liu Wei. Thanks for your advice on dissertation title selection and remind of do's and don'ts. Your knowledge and advice have great help in determining my dissertation direction. I would also like to thank Professor Mark Reynolds, Professor Tim French, and Professor Terry Woodings for discussing with me about the topic at the early stage of my research and helping me to sort through my thoughts.

I would like to thank all my classmates in UWA, who accompanied me when I was away from my parents in a foreign country. Thank you for discussing the details of my research with me and sharing your jokes, encouragement and understanding. It would be lonely at UWA without you guys.

The most special thanks go to my parents. Thank you for giving me unconditional trust and support through all this long process. You always told me to take care of myself, and never gave me any pressure. Finally I would like to thank my girlfriend, who is my only family in this foreign country. Thank you for giving me meticulous care when I felt stressed and cheering me up when I felt lost.

Thank you for all you guys who ever helped me. I will never forget your guidance and help.

Contents

List of Tables

List of Figures

CHAPTER 1

Introduction and Motivation

1.1 Introduction

A typical software project is a piece of complex work involving cooperation from many developers. With the growth of the number of developers, the potential interactions among the developers and the complexity of cooperation patterns grow quadratically [6]. When the software team grows bigger, especially when the team members cooperate with each other remotely, the communication cost will grow dramatically. The cooperation pattern and process among the developers, however, are neither obvious nor trivial for the project managers who are aiming to improve software quantity. Poor or high-cost interactions could both result in delay or even in the failure of the project. In this research, we focused on what developer social networks (DSN) look like and how to analyze and optimize the structure of the network to improve the efficiency of the cooperation among the developers.

Almost the only resource that we can use is the software repository of the projects. Fortunately, a majority of software projects are managed using by software repositories such as CVS and SVN. These software repositories contain the interaction information of the developers of the project and the evolutionary information of the project as well. All the data that we are interested in are contained somewhere in these software repositories. How to extract information that is of interest to us from the development process is the first challenge of our research. Specifically, we will crawl contribution information (Meta information) from the software repositories of different open source projects. Developer interaction patterns and characteristics are extracted and transferred into developer social network. With the information extracted from repositories, we will visualize and analyze the developer social networks of different projects. In a broader sense, software repositories include all resources that contain software development information such as version control system, email, and open source software website, e.g. SourceForge (http://sourceforge.com). While many researches focused on mining email systems [6] or open source websites [53], in this dissertation, we will focus on version control system (CVS and SVN) exclusively. There are three reasons that we choose version control systems: 1) Version control systems contain richest and most direct information of the project because they record the whole process of the project development; 2) The information of version control system for open source software are public available for most projects; 3) Almost all of large software projects use at least one version control system.

A developer social network is considered as a weighted graph with developers as its vertices and relationships between developers as its edges. It is a weighted graph because the relationships between developers have different weights denoting the strength of the ties. We will apply social network analysis techniques to these social graphs and discover intrinsic patterns that are interesting to us, with the aim of improving the cooperation of project team members. These patterns [12] extracted from the graph include but are not limited to:

1. **Degree distribution**: The degree of a node k, is defined as the total number of links connected to this node. Degree is a simplified measure of influence of a single node on other nodes.

2. **Diameter**: The longest distance of all the shortest paths between any two nodes. The shortest path between two nodes is defined as a path which passes fewest intermediate nodes among all the paths starting from one node and ending with the other node. Its value is the worst case of communication cost in the network.

3. **Cluster**: Lots of large scale social networks contain many subgroups which have strong ties among the group members within the subgroup and weak ties among group members from different subgroups. These subgroups are known as clusters which can be identified with their different centroids. Understanding the number and scale of clusters is of vital importance for the analysis of complex social networks.

4. **Clustering coefficient**: The clustering coefficient of a vertex in the network is defined as the ratio of the number of links started and ended at the node to the total possible number of links among its neighbors. And the clustering coefficient of the network is calculated by averaging the clustering coefficients of all its vertices.

5. **Centrality**: Developers who have more ties to other developers may be more important and active in a project. Centrality can be defined in different contexts, such as degree and betweenness.

6. **Temporal effects**: Most Social Network Analysis (SNA) researches treat the network as a static structure, but it is not true for developers' networks. On the other hand, the dynamic and evolvement of the networks are very important.

These measures, however, cannot capture all the characteristics that differentiate "healthy" (well-organized) developer social networks from the "problematic" (ill-formatted) ones if analyzed independently. Based on the measurements given above, it is possible to build a classification model to discriminate well-organized developer social networks from ill-organized networks using classification learning algorithms. Selected projects which are tagged as "healthy" or "problematic" manually will be our training and evaluation data. Based on these data, a classification model is learnt using some off-the-shelf classification methods and then the model can be used to generate predictions to more ongoing projects on whether there are problems with their developer social networks. These predictions will provide insightful alerts for project managers.

Furthermore, the system can also provide useful suggestions to improve software development process and efficiency as well. Possible recommendations include:

1. Module based recommendation:
   (a) Recommend developers to the modules according to their ability and interests
2. Developer based recommendation:
   (a) Recommend other developers
   (b) Recommend other modules

We consider developer-to-developer recommendation which chooses appropriate unconnected relationships in recommendation. We implement and compare two social network recommendation algorithms: Random Walk with Restart and Ant Colony Recommendation. The recommendation results are evaluated on real-world datasets.

The remaining of this dissertation will focus on the following three topics: 1) Construction and analysis of developer social networks; 2) Predicting project's "healthiness" using machine learning algorithms; 3) Recommendation in developer social network in order to optimize the network. Figure 1.1 presents the main topics that will be covered and their relationships. The data flow and modules

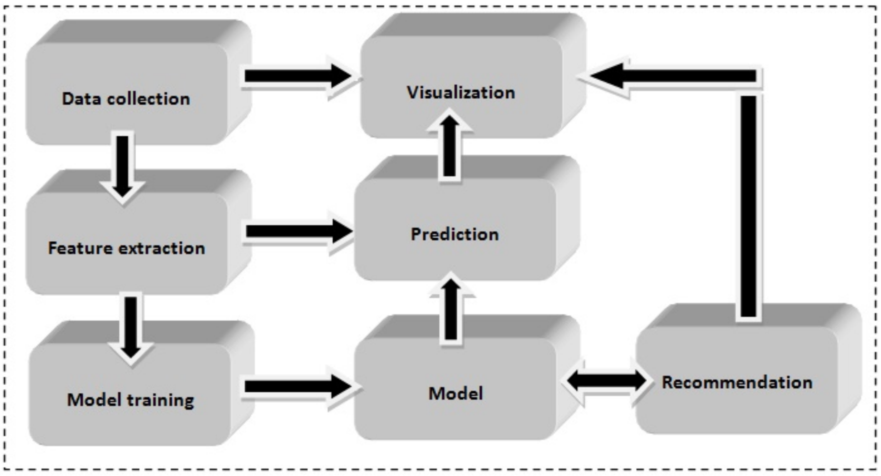


Figure 1.1: Data flow

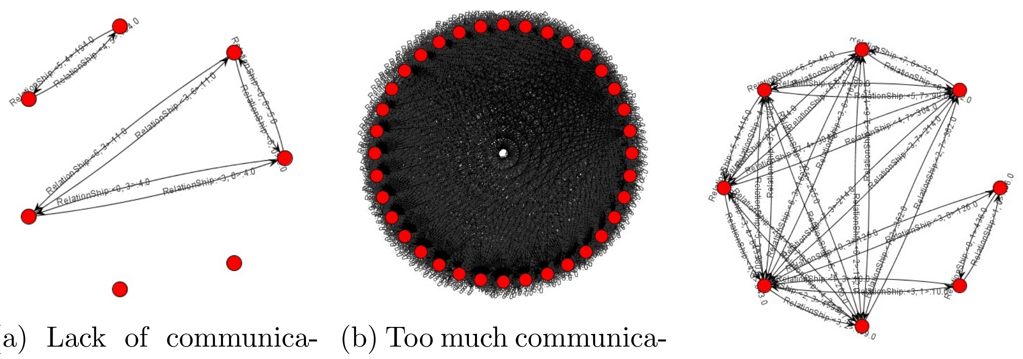
in our Developer Social Network Optimizing System (DSNOS) also follow this pattern.

After the data collection step, social network features will be extracted from the developer social networks which are constructed based on the data collected from software repositories. Machine learning algorithms are then applied on these features to fit manfully labeled projects as training data. Afterwards, the learnt model and the features extracted from new projects generate predictions of “healthiness” about the new projects. Meanwhile, we implement various visualization tools for better analyzing the data and presenting results in different steps, including the final recommendation module.

1.2 Motivation and Contributions

Motivation

Figure 1.2 shows three developer networks of three different projects. It is easy to see the problems with the first two networks. Intuitively, the first network does not have enough relationships among its developers which very likely indicates insufficient communications among its developers while it is just the opposite for the second network. James et al. [20] and Ye et al. [55] described the inefficiency brought by lack or overload of communication among team members. Compared with the other two types of networks, the third network is appropriate in terms of its network structure. Well-formed network structure will facilitate communication efficiency promoting the quality of the project while reducing the



(a) Lack of communication (b) Too much communication (c) Approriate

Figure 1.2: Three different developer social networks

communication cost to speed up the development. This observation motivates our research. We can build a system that judge the project from the aspect of its developer social network automatically. Moreover, we can quantify the importance of the factors that influence the healthiness of the network.

Contributions The contributions of this research are four-fold:

- We visualize and analyze social networks of open source software projects;
- Through investigating various projects, we found the most discriminating features of developer social networks for network healthiness prediction. Based on these features, various machine learning methods for classification are implemented and compared. And the cross-validation results validate the effectiveness of our prediction model.
- Based on the model trained above, we can tell whether the social structure of developers of a project is healthy or not, but a more important question is how to improve it. We take a step further to try to improve the healthiness of an ongoing software project by optimizing its developer social network. We make recommendations of the most valuable connections between the developers and the effectiveness of these recommendations is verified by both posterior facts (whether the recommendations are accepted by the developers) and our model.
- A developer social network system is implemented using Java providing the following functionalities: 1) Developer social network construction and visualization; 2) Social network healthiness evaluation and prediction; 3) Recommendation of developer connections aiming at optimizing the structure of current developer social network.

1.3 Thesis Structure

In Chapter 2, we give a thorough review of related research. First, we give the background of software repositories and the motivation of doing data Mining in Software Repositories (MSR) followed by a classification of the directions of MSR research. In each direction, we introduce the latest advances. Then, the definition and methodologies of Social Network Analysis (SNA) are introduced with special attention of their applications in the MSR settings. SNA researches in MSR are closely related to our work. Finally, we give a brief introduction of machine learning methods that we will use in the following chapters and their Weka implementations. We only include the most relevant materials to keep the description concise and to the point.

In Chapter 3, we apply machine learning methods to classify different developer social networks. We first describe how to construct a developer social network from the software repository data. Next we visualize the networks and analyze their topological features. Then machine learning methods are applied on these features to build classification models and finally the results and evaluations are presented.

In Chapter 4, we take a step further by recommending potential links in the network to optimize the network. We apply existing Random Walk method and a novel social recommendation algorithm based on Ant Colony algorithm to generate link recommendations in the developer social network. Finally, evaluations of the recommendations are reported.

Moreover, we have designed and built a Developer Social Network Optimizing System (DSNOS) providing functionalities described in the Chapter 3 and Chapter 4. The implementation details of the system are reported in Chapter 5.

Finally, conclusions and some future research directions are given in Chapter 6.

CHAPTER 2

Related Works

2.1 Software Repositories and Mining software repositories (MSR)

Software repositories contain most of the information throughout the whole process of software development [35]. Typical software repositories include source control systems, communications archives between project personnel, and defect tracking systems and project documentations. This research will focus on source control systems exclusively because they can be easily obtained for almost all open source projects and they provide the most plentiful and accurate information reflecting the status of a software project. Taking CVS (http://www.cvshome.org) for example, in addition to storing change patterns across different document versions, CVS also records project metadata such as version changes, modification comments, user names, modification timestamps, and every other changes. A typical software repository (taking CVS for example) usually includes the following two sections: administrative files stored in the root directory on server (known as CVSROOT), and detailed version control files (RCS file for CVS) associated with each code file and data file as well which store all the revision histories under their corresponding directory [35]. This information is a precious resource for analyzing both the current status and the evolution of the ongoing project with the aim of detecting possible risks and improving software development process.

With the fast growth of the complexity of software development, software repositories are providing more functionalities and flexibilities than ever before. They are acting as an important role in software development process nowadays, both for free software and commercial software. Furthermore, we have many off-the-shelf choices for managing our code and revisions of the code, such as CVS (http://www.cvshome.org), SVN (http://subversion.tigris.org/), Git (http://git-scm.com/), to name just a few. Applying data mining technologies on these voluminous data to improve software development is a rising and promising research area which is commonly referred to as Mining Software Repositories (MSR). The purpose of doing data mining in software repositories are many-fold: fault (bug) detection and resolving, developer performance appraisal, providing assistance to developers [59] and improving software design/reuse, etc. MSR is an emerging research area which is a joint research filed of software engineering and data mining technologies.

2.1.1 Objectives of MSR research

By utilizing Data Mining techniques (and many machine learning algorithms), Mining Software Repositories is an active research area which analyzes and experiments on historical data of software projects in order to better understand the software development and improve it in return. A deeper understanding of software repositories can assist us in guiding and enhancing the software development process and methods. The objectives of MSR research are multi-fold. Kagdi et al. [24] gave a survey about the taxonomy of this relatively new and attractive field. Although no formal classification of these objectives are available yet, these researches generally fall into the following three categories:

Evaluation and Observation

The most popular measure of the complexity of software projects and developers' work is "Line of Code" (LOC). However, the LOC measure fails to capture the structural complexity of a project and therefore may lead to an erroneous estimate of project complexity. Moreover, a team member's contribution to a project is appreciated not only in terms of how many lines of code he or she has written, but also in his or her influence on other team members. What is needed is to use data mining techniques to do a more fine-grained and flexible analysis of the project in order to improve the accuracy of estimation of both the project complexity and programmer's work. For example, Mierle et al. [35] proposed a machine learning method to evaluate programmers' performance based on the features extracted from a CVS repository. They tried logistic regression, Naïve Bayes and kNN methods (implemented in the Weka machine learning package), and achieved an average precision of 75%, with Naïve Bayes outperforming other methods.

Prediction and Risk Detecting

There are many uncertainties for an ongoing software project throughout its development process. How to detect defects and predict risks and changes in a project are central topics of software engineering. Guo et al. [18] employed Random Forest [9] learning methodology for predicting fault prone modules in a software system. Zimmermann et al. predicted further files to be changed by mining item coupling among source code files and functions [60]. They claimed their system can correctly predict 26% of future changes to files and 15% of precision in predicting changes in functions or variables. Menzies et al. [34] proposed a MSR method for defect detection. This kind of systems provide powerful assistance to detect risks in advance which is otherwise very hard to be found by code review.

Recommendation

Another major type of MSR methods is to recommend information to developers. There are two kinds of potential targets to be recommended in software projects: people and source code. As a relatively new research area in MSR, a few preliminary research results have been published, e.g. [23].

- **Similar developers**

Developers are not working alone in a project. There is always someone who can provide help or information to improve the development process. Recommendation algorithms can find other developers with similar interests as the current developer. More about developer recommendation will be covered in Section 2.2.5.

- **Similar files or modules**

Bouktif et al. [8] unveiled the relationships between source code files. After a developer makes modifications to a source code file, they recommend the related source code files to the developer that may also need to be changed.

2.1.2 Information Available in Software Repositories

In a broader sense, software repositories include all resources that contain software development information such as version control system, email, and open source software website, e.g. SourceForge (http://sourceforge.com). While many researches focused on mining email systems [6] or open source websites [53], in this dissertation, we will focus on version control system (CVS and SVN) exclusively. There are three reasons that we choose version control systems: 1) Version control systems contain richest and most direct information of the project because they record the whole process of the project development; 2) The information of version control system for open source software are public available for most projects; 3) Almost all of large software projects use at least one version control system.

As we have mentioned above, version control systems record the whole development process of a project. Any revision to the project, no matter how small it is, will be recorded in the revision histories which can be obtained easily. Take CVS for example again, revision histories about all transactions which and where files are actually modified, there is a corresponding RCS revision record inside the RCS file. All these records compose an interweaved revision table. This table tracks revisions, including information such as "diff" size and the modification timestamp. The information contained in these RCS files can be obtained by calling "*cvs log* $\$FILE\_NAME$" command which will get the revision histories of the code given by parameter "$\$FILE\_NAME$". Other version control systems provide similar information and log mechanism as well. The knowledge we can find in version control system include but not limited to the following questions:

1. Who had modified what;
2. Who collaborate with whom;
3. Who are more active than others;
4. Which module is more important;
5. How modules and files are related to each other;
6. How the software evolves over time.

MSR methods choose information of interests to analyze. In this research, we focus on "who collaborate with whom" information, which is best represented as a social networks composed of developers and their collaboration.

2.1.3 Methods

We categorize mainstream MSR methods into the following directions. All of these directions are covered in this research and recent advances in each direction are introduced as follows.

Visualization

Visualization methods visualize the structure (e.g., connectivity, degree, centrality and clustering effects) and evolution of the developer social network of a project to find interesting patterns that can provide assistance to project managers to improve the project. Storey et al. [44] gave a survey about existing visualization tools which do visualization of patterns mined from software repositories information. Weissgerber et al. [50] proposed three visualization techniques in order to examine programmers' collaborative patterns. These three methods are: "transaction overview", "file author matrix" and "dynamic author-file graph", the contribution and cooperation of developers are clear as well as the evolution of a project. Xie et al. [52] presented a new tool named CVSViewer3D which integrated functionalities of information extraction, processes, and visualization of CVS repositories. Visualization techniques are usually the first step in social network analysis because they provide intuitive information for further analysis introduced afterwards.

Statistics

Huzefa et al. [24] gave possible aspects of statistics of a project. For example, metrics like lines of code (LOC), number of developers, changing rates, number of bugs found, similarities among the code files and developer social network of a project can be calculated as features representing a project upon which some important observation and evaluation can be made. Several metrics for project quality evaluation are proposed by Norick et al. [39]. These metrics include "McCabe's Cyclomatic Complexity", "Lines of Code", "Comment Density" and "Maximum Nesting". Further, these features can be treated as the inputs of data mining methods to have a deeper understanding of the project.

Clustering

Clustering is a fundamental technique to analyse complex data in the data mining and machine learning area. As for MSR, clustering techniques are also very popular and they can be categorized into two categories: source code clustering and developer clustering. Clustering of either code files or developers is based on the fact that the source code files and developers have strong inner-group connections, respectively. Code clustering is to cluster code files according to their similarity and relationships. Developer clustering is to cluster developers according to their responsibilities and interests. Both of these clustering methods can give us a clearer view of the project to enhance our understanding of the healthiness project.

Prediction

Prediction is to reduce uncertainties based on the historical information. Usually, classification or regression models are built on the current observed data, and prediction is made upon this learnt model. Prediction is also one of the main topics in this thesis. Nagappan et al. [36] used Principle Component Analysis (PCA) on the code editing matrix and then implement their built regression model to predict potential defects of software projects. In [18], the random forest algorithm was employed to predict the robustness and fault-proneness of a given project.

Social network analysis

Social network analysis has a long history and it has become a very hot subject in this Internet era. Since our research method belongs to SNA, we will first give a detailed introduction of methods in Section 2.2.

Recommender systems

General recommendation strategies fall into two major categories: content-based and collaborative filtering [1]. Content-based algorithms give recommendation based on the semantic similarity among the source files and developers, while collaborative filtering algorithms recommend using collaborative information of the developers. Zimmermann et al. [60] used data mining technique to predict source code file co-change. After the relationship between code files has been found, similar code files will be recommended to the developer after the developer changes the source code file. Section 2.2 reviews another type of recommendation method: social network recommendation, in the context of developer social network analysis and more about social network recommendation can be found in Section 4.1 before we introduce our proposed recommendation algorithms.

Random Walk (RW) and Random Walk with Restart (RWR) algorithms are proposed for probability inferring problem large-scale complex networks. They are particularly suitable for solving recommendation problem in social network context, both for common social graph and bipartite social graph. The most well-known algorithm in the information retrieval area that employs random walk method is probably the PageRank algorithm proposed by Google for its webpage ranking algorithm. As we will see later, the classic PageRank algorithm can be viewed as a special impersonalized case of RWR [13].

A illustration of performing RW on graph is shown in Figure 2.1:

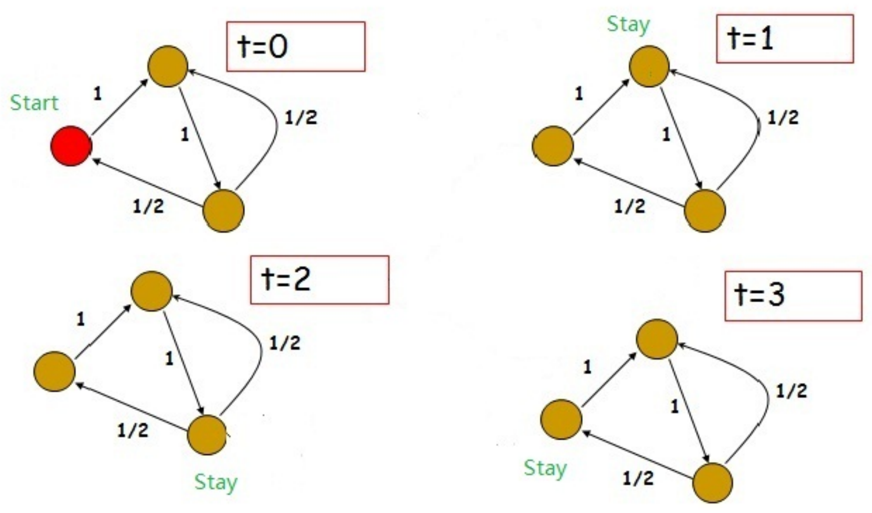


Figure 2.1: Illustration of random walks on graph

Yildirim and Krishnamoorthy [56] proposed a recommendation algorithm which performs Random Walks on a graph whose nodes denoting items and edges denoting the similarities between items. They evaluated their algorithm using data from MovieLens and validated RWR algorithm is effective for item recommendation, especially when the rating matrix is sparse. Hotho et al. [22] proposed a topic-specific random walk ranking algorithm and evaluated it on a dataset from del.icio.us, exploring folksonomies of bookmarks utilizing users' collaborative annotations. It outperformed adapted PageRank on their Folksonomy network which contains users, tags and webpages and their relationships.

Tong et al. [46] proposed a fast RWR algorithm. The essence of this approach is to utilize two important properties observed in many social graphs: a) linear correlations; b) blockwise and community-like local correlations. It is superior for large social networks. Another direction of handling large scale network is to parallelize the computation on clusters. Kang et al. [25] implemented parallelized RWR on hadoop clusters. Since the scale of developer social network is usually very small, computational efficiency is not our top concern. Therefore these large scale RWR algorithms are beyond the scope of this thesis.

Fouss et al. [14] took a step further. Other than transition probability, they used **average first-passage time** and **average commute time** measures to illustrate the usefulness of other Markov-based similarity measures on collaborative filtering. But there is no evidence that these measures perform better than transition probability in recommendation. The comparison will be interesting but beyond our scope.

Konstas et al. [27] showed that the RWR method is superior in recommendation precision than the standard Collaborative Filtering (CF) method for social music recommendation. Similar observations are also reported in [56].

2.2 Social Network Analysis (SNA) and Application to M-SR

Social Network Analysis (SNA) views social relationships of an interested domain such as friends, colleagues or even organizations, from graph theory aspect as a network consisting of **nodes** and **relationship**. Nodes (vertices, interchangeably afterwards) correspond to the individual actors in the network, and relationships (connections, edges and ties interchangeably afterwards) are the connections between the nodes. The objective is to understand the structure and the behavior within the social network, and increase the social benefits of the individuals in the network and the network as a whole.

2.2.1 Topological Network Properties

Topological properties of a social network are measurements of topological structure of the network. These properties (features, used interchangeably afterwards) represent how a social network is formed and the status of an individual node as well. Topological properties are an abstraction of the network structure based on which many interesting patterns and predictions can be generated. Moreover, topological measures constitute elementary features based on which more complicated models can be learnt from. We choose some common properties and give a brief introduction below.

Distance Centrality, DC [32]: is a measurement of a vertex's proximity to the other vertices in the network. It is also known as the closeness centrality. A higher DC value means that the vertex is averagely closer to the other vertices in the network. This measure is calculated on the vertex base. By averaging over all the nodes in the network, DC reflects the connectivity degree within the whole network. The well-known "Six degrees of separation" refers to the observation of the human connection network that "everyone in this globe is on an average approximately six steps away from others", i.e. the average DC of the human connection network is about 1/6.

DC is an important measure for the evaluation of individuates' importance in the network. Martinez-Romo et al. reported that the central employees (with higher DC) in the networks, they are tent to be a fast learner, better performer and contribute more to the organization [32]. Moreover, the vertex with higher DC has more influence on others than the ones with lower DC.

Betweenness Centrality, BC: The betweenness centrality of a vertex is defined as the ratio of the number of shortest paths between any other two nodes traversing the concerned vertex to the total number of shortest paths. BC can be interpreted as a measurement of the information flow importance [12].

Diameter: is defined as the longest of the shortest paths between any two nodes in the network [12]. It is a measure influenced by both the connectivity in the worst case and the scale (total number of nodes) of the network.

Cluster: A cluster in a social network is a collection of vertices that are more tightly connected with each other in the collection than with the vertices outside of the cluster [12]. Specifically, connectivity can be defined in many different aspects in the social network context, such as closeness, betweenness and local connection properties [31].

Clustering Coefficient, CC: The clustering coefficient is a measure that indicates the local connectivity among its neighbor vertices of a given vertex. The clustering coefficient of a vertex is calculated considering local triangle structures pertaining to the vertex, and the clustering coefficient of a social network is calculated by averaging all the clustering coefficients of the vertices [26].

Eigenvector Centrality, EC: EC is defined as the principal eigenvector (eigenvector corresponding to the highest Eigen value) of the adjacency matrix pertaining to the network [38]. This measure is sometimes better than direct measures such as closeness centrality and betweenness centrality especially when the network is sparse (average degree is relatively small). We will explain EC in more detail in Section 3.1.3.

The above measures are the most important and popular features of a social network based on which many other measures can be extended from.

2.2.2 Statistical and Machine Learning Models

Based on the measures listed above, statistical learning methods build more complicated models using these measures as their input features and these models usually can outperform any single feature in terms of their discriminant abilities.

Snijders et al. modeled the evolution of social networks and constructed Markov statistical models to simulate the transition probability of the networks

[43]. Markov chain Monte Carlo (MCMC) method [48] is also employed to learn the model.

Luis et al. [30] applied Social Network Analysis to open source software projects utilizing the Information in their CVS Repositories and gave several measurements and comparison of Apache, GNOME and KDE projects. In [47], simple machine learning models were built in order to model social behavior of web users based on social network data gathered from blogs, email lists and other web sites. These models were then used to simulate how users joined and left projects and make predictions about these activities. This work bears some similarity to our work, instead we use a version control system to extract social network information.

Xu et al. [54] discussed "small world phenomenon" and "scale-free" characteristics discovered in open source developers" network to have a better understanding of open source software development process. Yu et al. [57] proposed an approach to mine each developer's role in the team using CVS meta information. An interaction matrix denoting the committing interactions between any two developers is calculated and thus the core members of the team can be identified through matrix analysis methods. Christian et al. [6] performed an empirical study of the latent social structure of open-source teams through data mining in the developers' mailing list. Tang et al. [45] described an academia social network analysis tool: ArnetMiner. It extracted authors of academia papers and modeled simultaneously topical aspects of these papers, authors of the papers, and publication venues. It was an interesting attempt of applying topic model on social network data.

2.2.3 Temporal Effects

One of the most distinct characteristics of developer social networks from other type of social networks is in its evolutionary nature. In different phase of development process, the developer social network takes on different forms. Therefore, it is important to analyze the change and evolvement of the network besides its static features.

Robles et al. [40] provided the changes of main measurements such as "Number of commits", "Number of developers" and "LOC" of typical software projects over time. Barabasi et al. investigated the change pattern of different measurements such as degree, closeness and clustering coefficients along with the evolution of scientific social networks [5]. A simple linear model is proposed to model and simulate the evolution of social network. Gao et al. [16] investigated the structure and the dynamical factors that influence the evolution of developer social networks. The evolutions of both the software projects and their developer social networks are remarkable and usually follow certain patterns.

We also incorporate temporal features into our classification and recommendation models.

2.2.4 Social Influence Analysis

There are always some key roles in developer social networks who have much impact on both the project and other developers. It is important to quantify this influence to have a better understanding and control of the project. Xu [53] categorized OSS developers into the following groups. Obviously, the importance and contribution of these developers are listed in descending order.

1. **Project leaders**: who are responsible for the design and coordination globally;
2. **Core developers**: who frequently contribute to projects and are responsible for main modules;
3. **Co-developers**: including peripheral developers who contribute to the projects less frequently than core developers;
4. **Active users**: who have other types of contributions except committing code.

In this research, we only consider the top three types of developers who directly modify the source code of the project. Moreover, there could be finer-grained classification of the developers in a project according to his or her influence on the project. For example, in [29], Liu et al. provided a method quantitatively learning influence between users and how influence spreads in heterogeneous networks using graphic model. Generally speaking, influence is a complex and invisible force that sometimes determines social network dynamics and user behavior within the network. However, influence can be partially represented by measurements such as degree and closeness implicitly, therefore we won't consider influence as a stand-alone factor in this research.

2.2.5 Social Network Analysis for Recommendation

Based on the analysis of current and past developer social networks, recommendations could be made to improve the current social network to facilitate the development which is also one of the main objectives of this dissertation. We mentioned some general recommendation algorithms in Section 2.1.3; here we focus on recommendation using social network analysis method. Carmagnola et al. [11] proposed SoNARS algorithm, a social network-based algorithm which targeted users as members of social networks and can be fused with classical collaborative filtering and content-based approaches. And the overall trend of the network is computed as well as the user interests in order to provide users with recommendations which consider both the trend of their network and their personal interests.

Yu et al. discussed three strategies for recommendation in social networks: content-based, collaborative filtering and influential ranking [58]. The first two are classic recommendation strategies in the recommender system, while the last one is unique in social network context which recommends people according to various criteria including the number of followers, Friends of a Friend (FOAF) etc. Hahn et al. [19] investigated whether prior collaborative relationships impact open source software (OSS) development team formation and developers' joining behaviors. The conclusion of their investigation is that more prior collaborative relationships in the developer social network probably indicate a project will absorb more developers. This discovery can be utilized for a simple yet effective collaborator and project recommendation.

2.3 Developer Social Networks

Sufficient and efficient communications among team members are encouraged and important for the success of a software project. However, the results and effectiveness of communications are hard to evaluate and quantify. In MSR jargon, these communications can be modeled as collaborative social networks among developers. Developer social networks have their own characteristics like their high dynamics. A healthy developer social network keeps evolving while a morbid network usually stays static or in chaos. Since its significant influence on the success of the project, can we tell a good network from bad ones before it is too late to change something? This thesis aims to give an initial answer. Specifically, there remain two important questions to be answered: 1) Is the current developer social network suitable for this project and does it facilitate or hinder the progress of the project? 2) How to improve the current developer social network to improve the communications among developers and therefore increase development efficiency and dodge unnecessary traps. These two topics which are interrelated are the main focus of this research.

The research dedicated for software developer social network is relatively sparse. It is a joint field of social network and software engineering. Developer social network has its own properties and it is important for facilitating the development process and improving software quality, it deserves special attentions from both academia and industry. We list some important characteristics of social networks formed by developers compared with other types of networks below:

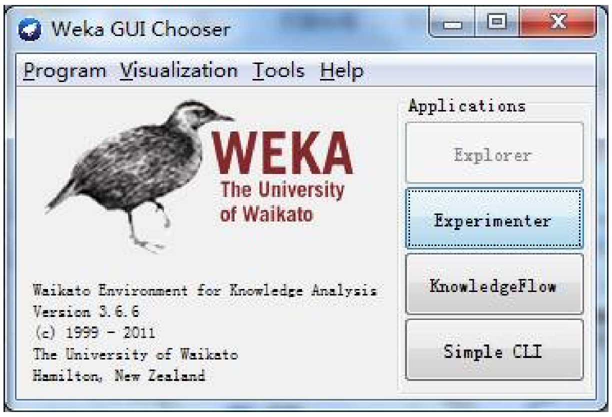
- Objective: the objective to analyze developer social network is to facilitate the development process rather than focusing on the utility of the individuals in the network.
- Bipartite graph nature: A developer social network consists of developers and code files. Thus it is a bipartite graph (developers as one group and source code files as another group) in essence although it can be mapped to a common graph with some loss of information.
- Dynamics: A developer social network of a typical project evolves quickly and the evolutionary process contains important information which cannot be obtained from static network features [53].

2.4 Machine Learning with Weka

Machine learning is referred to as a bunch of computational methods that learn statistical models of given data and to find useful patterns underlining the data which is often used interchangeably with the term “data mining”. Actually, MSR is to apply machine learning methods to mine interesting patterns in software repository data. It is impossible to give a complete introduction to machine learning methods here; we refer the readers to the book [7] for more details. In the following paragraphs, we will give a brief introduction to the machine learning algorithms used in our experiments, i.e. decision tree with C4.5, logistic regression and Naïve Bayes. For experiments, we used Weka [51], an open source implementation of these algorithms.

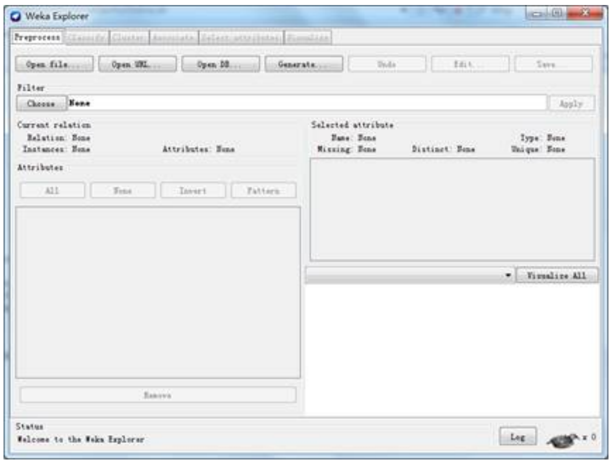
2.4.1 Weka

Weka is an open source package implementing various off-the-shelf machine learning algorithms for data mining tasks [51]. The machine learning tools in the Weka library can either be applied directly on a dataset or called from our own Java code which can also do any customizations and modifications. Weka also provides powerful tools for data pre-processing, classification, regression, clustering, visualization and evaluation. Figure 2.2 and Figure 2.3 below show the main GUI interface of Weka. In our project, Weka java library is integrated and called as a library of the project. We will use Weka to implement all the algorithms mentioned in Section 3.2 and compare their performances on our task.



Waikato Environment for Knowledge Analysis
Version 3.6.6
(c) 1999 - 2011
The University of Waikato
Hamilton, New Zealand

Figure 2.2: The main interface of Weka: provides four functionalities



Welcome to the Weka Explorer

Figure 2.3: The GUI interface of Weka explorer

2.5 Conclusions

Three domains of related works have been introduced in this review: Mining software repositories (MSR), social network analysis (SNS) and machine learning with Weka. Although much work has been done as we have seen in the survey, MSR is a relatively new research field and the research works are significantly fewer than the other two domains in terms of number of publications. Moreover, the general depth and width of MSR research are still not comparable to the other two fields yet. This is why this topic is of special interest to us. Machine learning as a powerful tool, its importance in MSR area becomes aware by more and more researchers. However, we see very few literatures (e.g. [5]) which apply SNS on developer social network from a machine learning perspective. We will describe our method in Chapter 3. Furthermore, recommendation in developer social network is a very new problem. Our solution will be described in Chapter 4.

CHAPTER 3

Developer Social Network Analysis and Modeling

In this chapter, we will first construct developer social networks from different projects and visualize them. Then we will build classification models for judging and diagnosing the status of a given developer social network. Detailed experimental results are reported at the end of this chapter.

3.1 Data Collection and Analysis

3.1.1 Developer Social Network Construction

Our training and testing data are both from open source projects which can be easily collected from Internet. We chose 30 projects from several open source hosting websites. The projects' source code and their change log are both crawled from their corresponding repositories respectively, which is managed by either CVS or SVN.

The data collection process takes the following steps:

1. Choose a project and find the URL of the code repository (SVN or CVS) for this project;
2. Check out the source code;
3. Get the revision history (change log) for each of the qualified source code (belongs to predefined file type set);
4. Generate the DSN file;
5. Construct the network using the DSN file;
6. Visualize the network and network parameters;

7. Perform network clustering;

8. Label the project.

Among some of the steps, our design involves many tricks and details that we explain below.

Choose a project

We chose these projects according to the following criteria:

- The project has lasted for more than one year
- The project has more than (including) 3 developers
- The project belongs to one of the following four categories: C or C++ project, Java project, PHP project or Python project.

Source code check out

We check the project out by calling the system command "*svn checkout*" or "*cvs checkout*" for SVN project and CVS project respectively in Java. The project code will be saved in the local directory.

Change log crawling

The change history log of the code files is also crawled from software repository to local directories for further processing. Similar to source code check out, the change log can be crawled by calling system command "*svn log* $\$CodeFilePath > \$LocalPath$" or "*cvs log* $\$CodeFilePath > \$LocalPath$" for each of the source code.

The following is a snippet of change log of a code file of a SVN project. It contains three change records of this file. We concentrate on the developer and modify time information in each record.

```
------------------------------------------------------------------------
r682 | zlatkarp | 2010-06-26 15:41:10 +0800 (Sat, 26 Jun 2010) | 3 lines

Add pre-processing for PNG files in order to resolve
issues with transparency.

------------------------------------------------------------------------
r464 | krokodil | 2009-02-16 06:34:25 +0800 (Mon, 16 Feb 2009) | 2 lines

command line script for debugging

------------------------------------------------------------------------
r463 | krokodil | 2009-02-15 14:52:45 +0800 (Sun, 15 Feb 2009) | 3 lines

more sophisticated parameters support.
```

One thing should be clarified here is that we only care about source code files that implement certain functions when crawling their change log, i.e., documentation, configuration and log files are not included. The file formats that we are interested in include: ".c", ".cpp", ".cc", ".java", ".php", ".asp(x)", ".sh", ".py", ".pl", ".js", and ".html".

After the change logs are saved, the logs are processed into format called "DSN" that is more convenient for further analysis. There is one and only one DSN file for each project. A snippet of a DSN file is shown below.

```
EditInfo [DEVELOPER="krosenvold", CODEFILE="/maven-3_log/trunk_maven
-artifact_src_main_java_org_apache_maven_artifact_DefaultArtifact.java",
TIME="Tue Feb 01 23:29:32 CST 2011"]

EditInfo [DEVELOPER="bentmann", CODEFILE="/maven-3_log/trunk_maven
-artifact_src_main_java_org_apache_maven_artifact_DefaultArtifact.java",
TIME="Wed Aug 25 06:46:07 CST 2010"]

EditInfo [DEVELOPER="hboutemy", CODEFILE="/maven-3_log/trunk_maven-
artifact_src_main_java_org_apache_maven_artifact_DefaultArtifact.java"
TIME="Sun Jun 27 07:16:18 CST 2010"]
```

Network constructing and basic analysis

The DSN file of the project is used for constructing the developer social network for the corresponding project. We extract data structures named **Developer** and **Relationship** from the DSN file. Each **Developer** corresponds to a developer who contributes to the project. Each **Relationship** is defined when two developers have co-edited the same code file. The relationships represent the cooperation between developers and therefore represent the coupling or dependency among the developers. The developer and relationship correspond respectively to the vertex and edge of the graph representation of the developer network. The edges may have different weights because developers spend different time and effort on different code files. Specifically, the weight of the edge between two developers is the sum of their change frequencies (commit times) of all the co-edited files. Therefore, our DSN graph is a weighted graph with the weight of edge representing the relative strength of their connections.

The intuition is that a "healthy" network contains moderate density of relationships. Because too few relationships among developers indicate the project lacks cooperations among the developers while too many relationships among developers usually indicate the module division is not clear and rational so that communication costs among the developers would be high. Other than the density of relationships in the network, other measures such as betweeness and diameter of the graph can also reflect the current status of the network. Some of the advanced features will be explained in the next two sections.

3.1.2 Developer Social Network Visualization

Visualization is an important tool for social network optimization. Visualization can give us perceptual knowledge about the status of a given network. More importantly, it is the first step of feature selection. Through the visualization of different types of networks, it is easier to find more good features which will improve the model performance. All the prediction and recommendation results generated by our model are easier to be understand and accepted by the project managers through visualization. Therefore, we do visualization before and after our model training and recommendation process. We will see the visualization results in experiment parts of this and the next chapters.

3.1.3 Developer Social Network Clustering and Adjacency Matrix

In this and the next section, we will introduce some complex types of social network features, i.e. clustering and Eigen vector related features, in more detail.

Clustering

Due to the division of different modules of a software project, developers naturally form subgroups in which the members are highly connected, while sparsely connected even separated among different subgroups. This phenomenon is also

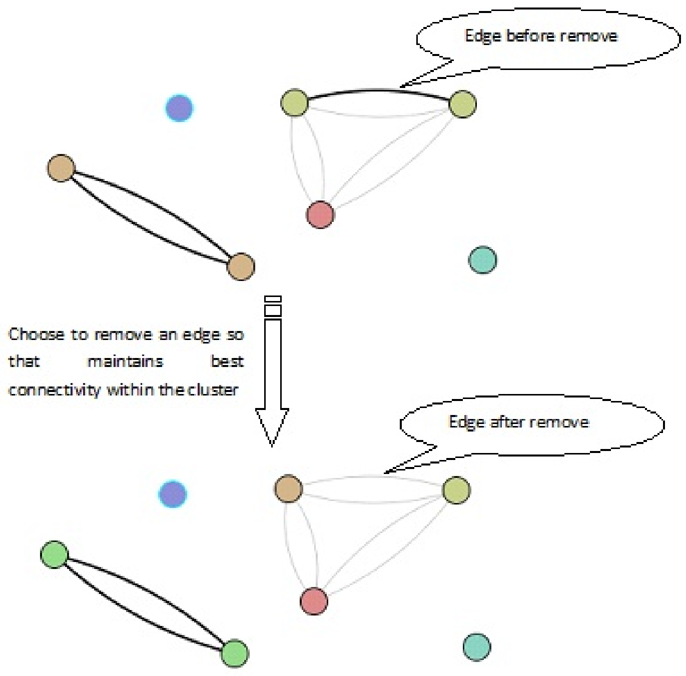

Edge before remove
Choose to remove an edge so that maintains best connectivity within the cluster
Edge after remove

Figure 3.1: Illustration of removing an edge for clustering

observed in other social networks which is known as clustering. Properly formed cluster structures are of vital importance for the success of a project.

There are many strategies for performing clustering in a graph. We follow the clustering algorithm using betweenness criterion which computes clusters for a graph based on the betweenness property of the edges.

Figure 3.1 shows the process of the edge betweenness clustering. It removes edges from the graph following the rule that keeps the integrity of connection in a cluster while removes the connections between different clusters.

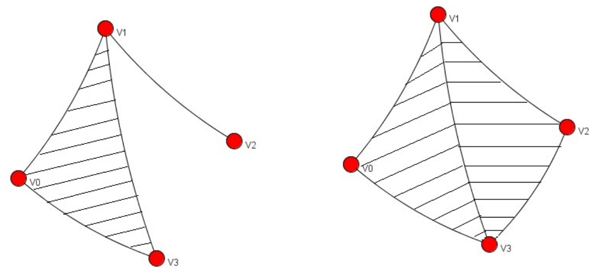
Clustering Coefficient

Clustering coefficient is a measure telling that whether vertices in a graph tend to cluster together or not [37].

$$clust(i) := \frac{t}{N_i(N_i - 1)/2} \tag{3.1}$$

where t is the number of triangles (dash area in Figure 3.2) including node i and N_i is the number of direct neighbors of node i.

Figure 3.2 shows two simple networks that both contain four vertices. The



(a) Four connections (b) Five connections

Figure 3.2: Two simple social networks

Clustering Coefficient score of each vertex is shown in Table 3.1. On average, network (b) has a higher clustering coefficient than network (a) which means the vertices in network (b) are clustered closely than in network (a). This conclusion agrees with our observation.

Network	V0	V1	V2	V3	Average
a	1	1/3	0	1	7/12
b	1	2/3	1	2/3	5/6

Table 3.1: Cluster coefficients of the illustrated network

3.1.4 Matrix Representation and EigenVector

For the convenience of further analysis, the network can also be represented by a matrix with each of its elements a real number representing the relationship weight between any developer pairs. This matrix is known as **Adjacency matrix** of a graph in social network analysis field.

EigenVector Centrality

EigenVector is an important attribute of a matrix. We will introduce the definition of Eigen Vector and its application to calculate Eigen centrality feature. For a given adjacency matrix A, its EigenVector x is defined as follows, where λ is a real number:

$$\lambda \mathbf{x} = \mathbf{A}\mathbf{x} \tag{3.2}$$

There are many different Eigen vectors x for each adjacency matrix A. However, we only care about the highest Eigen value which results in the desired centrality measure. The Eigen vector corresponding to the highest eigenvalue is the centrality vector of the graph where the i^{th} element is the Eigen Centrality (EC) score of the i^{th} node in the network.

We will revisit this important concept in the feature extraction section and recommendation section in the later chapters.

3.1.5 Project Time Slicing

In order to capture the dynamics of the project development and increase the number of training instances, we adopt a project time slicing strategy. For each project, we capture 10 snapshots in 10 different timestamps of its DSN status to form 10 similar but different instances for training.

The timestamp of the first commit of the project is denoted as *Starttime* and the timestamp of its last commit is denoted as *Endtime*. We divide the period from *Starttime* to *Endtime* into 10 sub-periods, evenly. At the end of each sub-period, we analyze the project social network and calculate its features. Therefore, for one project, we will get 10 related but different training instances.

3.1.6 Project Labeling

Criteria: Basically, we use two criteria to predict a project as healthy or unhealthy, project result criterion and social network criterion.

- Project result criterion (40%): if a project is active and has fair influence (awareness and the number of downloads) on the society, we are inclined to tag it as healthy; otherwise, it is problematic.
- Social network criterion (60%):
  - Average Degree: a project with the average degree not too big nor too small and the degree variance is not too big is considered as healthy;
  - Connectivity: the network is fully connected, without or with few isolated developers is considered as healthy;
  - Key developers and peripheral developers: how many key developers and peripheral developers the network has, less is more healthy;
  - Clustering coefficients: removal of how many edges will make the network unconnected, the more, the better.

Project Name	Website	Repository Type	Number of Developers	Number of code files	Time Slice	Development Duration
				15	1	2008-07-04 ∼ 2008 − 11 − 18
				22	2	2008-11-18 ∼ 2009 − 04 − 04
				27	3	2009-04-04 ∼ 2009 − 08 − 19
				35	4	2009-08-19 ∼ 2010 − 01 − 03
				41	5	2010-01-03 ∼ 2010 − 05 − 20
				66	6	2010-05-20 ∼ 2010 − 10 − 04
GoogleTest	GoogleCode	SVN	8	96	7	2010-10-04 ∼ 2011 − 02 − 18
				102	8	2011-02-18 ∼ 2011 − 07 − 04
				119	9	2011-07-04 ∼ 2011 − 11 − 18
				122	10	2011-11-18 ∼ 2012 − 04 − 03
				102	1	2009-04-02 ∼ 2009 − 07 − 22
				155	2	2009-07-22 ∼ 2009 − 11 − 11
				218	3	2009-11-11 ∼ 2010 − 03 − 03
				320	4	2010-03-03 ∼ 2010 − 06 − 23
				399	5	2010-06-23 ∼ 2010 − 10 − 13
				407	6	2011-02-02 ∼ 2011 − 05 − 25
chromium	SourceForge	SVN	58	532	7	2011-05-25 ∼ 2011 − 09 − 14
				587	8	2011-09-14 ∼ 2012 − 01 − 04
				645	9	2012-01-04 ∼ 2012 − 04 − 26
				708	10	2012-04-26 ∼ 2012 − 04 − 26

Table 3.2: Project Details

For instances sliced from the same project, their labels are forced to be the same, i.e. all the instances from the same project are labeled “healthy” or “problematic”.

A sample of sliced and labeled projects is listed in Table 3.2, there are two projects which are sliced into 20 instances. One project is labeled as “healthy” and the other one is labeled as “problematic”.

3.2 Model Training and Experiments

In this section, we present our training methods and make comparison of discriminate models for project developer social network classification and diagnosis.

3.2.1 Feature Explanation

Before we continue, the notations are listed in Table 3.3 for the convenience of explanation.

Notations	Explanations
$\mathcal{G}$	Graph
$\mathcal{V}$	Vertices (developers) set
$\mathcal{F}$	Code file set
$\mathcal{E}$	Edits set, with e_{vf} meaning developer v edits file f
ε	Edges (relationships) set
n	Number of vertices (developers)
ℓ and $\ell_{(t)}$	Number of edges (relationships) and number of edges at time (t)
m	Cluster number
ℓ^-	Number of removed edges for clustering
d_i	Degree of vertex i
t_v	Number of triangle structures attached to vertex v
$\mathcal{N}_v$	Number of neighbors linking with vertex v
g_{st}	Number of shortest paths linking vertex s and t
g^i_{st}	Number of shortest paths linking vertex s and t which contain vertex i
$\rho(G)$	Assortativity of graph G

Table 3.3: Notations

We choose from numerous candidate social network features that are most representative of the topological features of the graph.

- Degree Centrality (DC): the degree (including in-degree and out-degree for directed graph) of the vertices. In-degree is the number of inward edges of a vertex (edges ended with the vertex) and out-degree is the number of outward edges from a vertex (edges started from a vertex) in a directed graph [12].

- Betweenness Centrality (BC): Betweenness Centrality is a measure of occurrence of a given vertex as a node in the shortest path between any other two vertices [15]. $bc_i = \frac{\sum_{s<t} g_{(st)^i}/g_{st}}{\frac{1}{2}n(n-1)}$, where s and t are any two vertices and g_{st} is defined as the number of shortest path between them. From the definition, it is obvious that vertex with higher BC value is more important because more other vertices depend on it to reach other vertices (through their shortest path).

- Closeness (CS): is defined based on the vertices. The CS of a vertex is the "inverse of the average length of all the shortest paths from this vertex to all the other vertices in the graph" [12]. If there is no (directed) path between the two vertices then the closeness between the two vertex is defined as $1/n$.

- Assortativity Coefficient (AC): is a measure of extent to which vertices of similar degree would connect to each other [37]. $\rho(G) = 1 - \frac{\sum_{i\sim j}(d_i - d_j)^2}{\sum_{i=1}^{n} d_i^3 - 1/2\ell(\sum_{i=1}^{n} d_i^2)^2}$, where $i \sim j$ means vertex i has an edge to vertex j. If $\rho > 0$, the graph is assortative and is disassortative if $\rho < 0$.

- Diameter (DM): is defined as the longest path among all the shortest paths between any two vertices [12]. If there is no path between two vertices, then the diameter of the graph is defined as n.

- Cluster Number (CN): is defined as the number of clusters of vertices if a certain number of edges is removed. The edges are removed to maintain best connectivity within the clusters.

- Cluster Coefficient (CC): CC of a graph is defined as the arithmetic mean of CC of all its vertices defined in Equation 3.1 as follows: $CC := \frac{\sum_{v\in V} \frac{t_v}{|\mathcal{N}_v|(|\mathcal{N}_v|-1)/2}}{n}$ [37].

- Density (DS): the density of edges in the graph, ℓ/n [12].

- EigenVector Centrality (EC): The Eigen Vector corresponding to the highest Eigen Value of the Adjacency matrix.

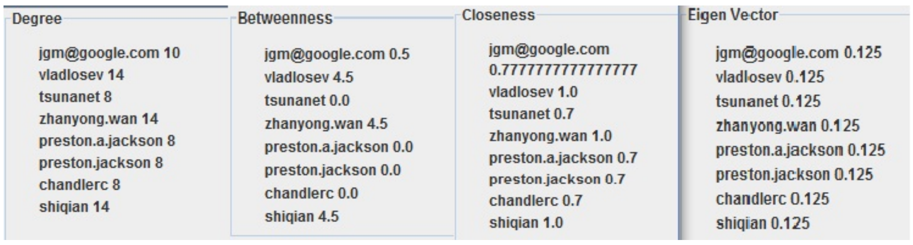
- Edge Increment (EI): The increment of number of edges from snapshot of the last timestamp divided by current number of edges.

Some of the features are calculated on the vertex basis, i.e., we will get n values for each of these feature where n is the number of vertices in the graph. However, we concern on the graphical characteristic of a social network as a whole instead of a single vertex in the graph. Therefore, we calculate two summary statistics for each of these features: *mean* and *variance*. *mean* is the arithmetic mean of the values of the feature for all vertices and *variance* is the standard variance of the values of the feature for all vertices. A complete list of the features are listed in Table 3.4, each row corresponds to a type of feature. And these features are classified into two types: graph-based features which are calculated on the graph and vertex-based features which are calculated for each vertex in the graph.

Figure 3.3 shows the features calculated on the vertex base of the social network of "GoogleTest". There are 8 developers in the network. The features are (from left to right): degree, betweenness, closeness and Eigen Vector. These are the feature values of a "healthy" network.

Feature name	**Explanation**	**Type**
normalized_degree	$\frac{2\times\ell}{n\times(n-1)}$	Graph
degree_mean	The average of degrees over all vertices	Vertex
degree_variance	The variance of degrees over all vertices	Vertex
betweenness_mean	The average of betweenness over al- l vertices	Vertex
betweenness_variance	The variance of betweenness over al- l vertices	Vertex
normalized_closeness	$closeness\_mean/n$	Vertex
closeness_mean	The average of closeness over all ver- tices	Vertex
closeness_variance	The variance of closeness over all ver- tices	Vertex
assortativity	AC	Graph
diameter	Longest path among all the shortest paths between any two vertices	Graph
normalized_diameter	$diameter/n$	Graph
cluster_1_4	Number of clusters (CN) when 1/4 edges are removed from the graph	Graph
cluster_2_4	Number of clusters (CN) when 2/4 edges are removed from the graph	Graph
cluster_3_4	Number of clusters (CN) when 3/4 edges are removed from the graph	Graph
cluster_1_4_variance	Variance of cluster sizes when 1/4 edges are removed from the graph	Graph
cluster_2_4_variance	Variance of cluster sizes when 2/4 edges are removed from the graph	Graph
cluster_3_4_variance	Variance of cluster sizes when 3/4 edges are removed from the graph	Graph
cluster_coefficient_mean	Mean value of "CC" for all vertices	Vertex
cluster_coefficient_variance	Variance of "CC" for all vertices	Vertex
density	"DS"	Graph
eigen_mean	Mean value of "EC" for all vertices	Graph
eigen_variance	Variance of "EC" for all vertices	Graph
edge_delta (EI)	$\frac{\ell_{(t+1)}-\ell_{(t)}}{\ell_{(t+1)}}$	Graph

Table 3.4: Feature list


```
Degree
    jgm@google.com 10
    vladlosev 14
    tsunanet 8
    zhanyong.wan 14
    preston.a.jackson 8
    preston.jackson 8
    chandlerc 8
    shiqian 14

Betweenness
    jgm@google.com 0.5
    vladlosev 4.5
    tsunanet 0.0
    zhanyong.wan 4.5
    preston.a.jackson 0.0
    preston.jackson 0.0
    chandlerc 0.0
    shiqian 4.5

Closeness
    jgm@google.com
    0.7777777777777777
    vladlosev 1.0
    tsunanet 0.7
    zhanyong.wan 1.0
    preston.a.jackson 0.7
    preston.jackson 0.7
    chandlerc 0.7
    shiqian 1.0

Eigen Vector
    jgm@google.com 0.125
    vladlosev 0.125
    tsunanet 0.125
    zhanyong.wan 0.125
    preston.a.jackson 0.125
    preston.jackson 0.125
    chandlerc 0.125
    shiqian 0.125
```

Figure 3.3: Some of the features of the GoogleTest project's developer social network

3.2.2 The Classification Algorithms

Similar to feature selection, we have even more candidate machine learning models to try. It is impossible and unnecessary to do an exhaustive exploration. We finally choose the following 6 models for a comparison.

- Decision tree
- Random forest
- Logistic regression
- Linear regression
- Naïve Bayes
- Support vector machine

Decision Trees: C4.5 and CART

Decision trees model is to build a "tree" model to represent the training data and prediction is made upon the "tree". C4.5 is an algorithm which generates a pruned (removing unnecessary nodes from the tree) decision tree developed by Ross Quinlan and Classification and Regression Tree (CART) is another decision tree algorithm proposed by Breiman [10].

Amongst numerous data mining methods, decision trees have the following advantages [41]:

- Very simple to understand and interpret. People are able to interpret decision tree models after visualization and can optimize them manually.

- Decision trees can accommodate various types of data. Therefore, little data pre-processing work is needed. Many other techniques often require data pre-processing techniques such as data normalization, data discretization, creating dummy variables and removing null values.

- Robust to noisy data. Decision tree models handles noisy data which are ubiquitous in the training set elegantly.

- Decision tree models are designed accompanied with feature selection ability which is very important for data mining algorithms.

Logistic Regression

Logistic Regression (LR) is actually a model for classification. It is one of most powerful models for binary classification (classify the instance into one of the two categories) [21]. Unlike many other machine learning algorithms, a global minimum can be found easily by gradient descent method instead of many local minima which is preferable for machine learning problems.

Naïve Bayes

As one of the oldest and simplest methods for machine learning, Naïve Bayes [7] outperforms many more advanced and complex algorithms in many competitions. Therefore, it is still one of the widest used machine learning algorithms. The algorithms we have chosen have one characteristic in common: good interpretability. This is of vital importance for our target: to interpret and to improve developer social networks.

Random Forest

Random forest [9] is a bagging algorithm that ensembles a bunch of classification or regression trees. After the training process of each of the ensemble trees, prediction can be made by combine (averaging for regression task and majority vote for classification task) the prediction results of the ensemble of the trees which is actually a standard bagging process. With the help of bagging, random forest obtains a significant performance improvement over any base tree classifier either CART or C4.5 as its base learner. Our experiments also validate this observation.

Support Vector Machine (SVM)

SVM is the state-of-the-art model verified by its performance in many important competitions. For classification tasks, especially for small-scale data, SVM usually generates the best results among many candidate models [7]. Compared with other models, SVM has two advantages: 1) it can deal with non-linearity in feature space by introducing kernels; 2) it maximizes the margin of the classifier to guarantee the performance on test dataset. We will not talk about the implementation details of SVM. The readers are referred to Chapter 6 and Chapter 7 of [7] and the implementation of SVM in Weka which we use for experiments.

3.2.3 Evaluation Methods and Measures

For a classification task, the evaluation measurements are standard and comparable. We use **Precision**, **Recall** , **F-measure** and **AUC** measurements detailed below.

First, we give definitions of different combinations of classification results and real categories as follows.

Category		**Real category**	
		Yes	**No**
Classifier results	**Yes**	TP (True Positive)	FP (False Positive)
	No	FN (False Negative)	TN (True Negative)

Table 3.5: Different combinations of classification results

Precision

Precision is the percentage of real problematic projects within the projects which are classified as "problematic", while recall is the percentage of problematic projects within the projects which are classified as "problematic" to all the real problematic projects [42].

$$Precision = \frac{TP}{TP + FP}$$

Recall

$$Recall = \frac{TP}{TP + FN}$$

F-measure

We pursue higher Precision and higher Recall simultaneously. Unfortunately, Precision and Recall measures are often contradicting. $F - measure$ is defined to generate a comprehensive evaluation.

$$F - measure = 2 \times \frac{Recall \times Precision}{(Recall + Precision)}$$

Area Under Curve (AUC)

Classification AUC is a measure considers both precision and recall. It is defined as the area under the ROC (Receiver Operating Characteristics) curve [7]. It is a positive real number which is equal or less than 1. The bigger AUC is, the better. Detailed AUC formula is given by Equation 4.5.

$10 - fold$ **Cross Validation**

Cross validation is an important technique for machine learning result evaluation by increasing the confidence of performance confirmation. By partitioning (equal allocation in general) all the instances into mutual exclusive and complementary subsets, training a model based some of these subsets (training set), and validates the model on the other subsets (validation set). To reduce variability, we use $10-fold$ cross validation that applies 10 rounds of cross-validation process using different partitions for each round, independently. Therefore, each sample has been included as training sample for 9 times and as validation sample for 1 time.

One thing needs to be clarified here is that the instances from the same project are partitioned into the same subset. Otherwise, the similar instances may both appear in training and evaluation datasets and therefore will affect the final performance evaluation.

3.2.4 Results and Discussions

Model	**Precision**	**Recall**	**F-Measure**	**AUC**
Decision Tree	0.691	0.675	0.669	0.779
Naïve Bayes	0.600	0.585	0.550	0.782
Linear Regression	0.544	0.555	0.523	0.586
Logistic Regresssion	0.656	0.642	0.627	0.778
Random Forest	**0.785**	**0.755**	**0.745**	**0.942**
SVM	0.537	0.555	0.491	0.600

Table 3.6: Model performance

We also plot the ROC curve in Figure 3.4. The X-axis is the ratio of instances that are classified as "problematic" and the Y-axis is the ratio of recalled real "problematic" instances in all real "problematic" instances. The area under the curve is our AUC measure. We can see that the bigger is AUC, the better is the classification model.

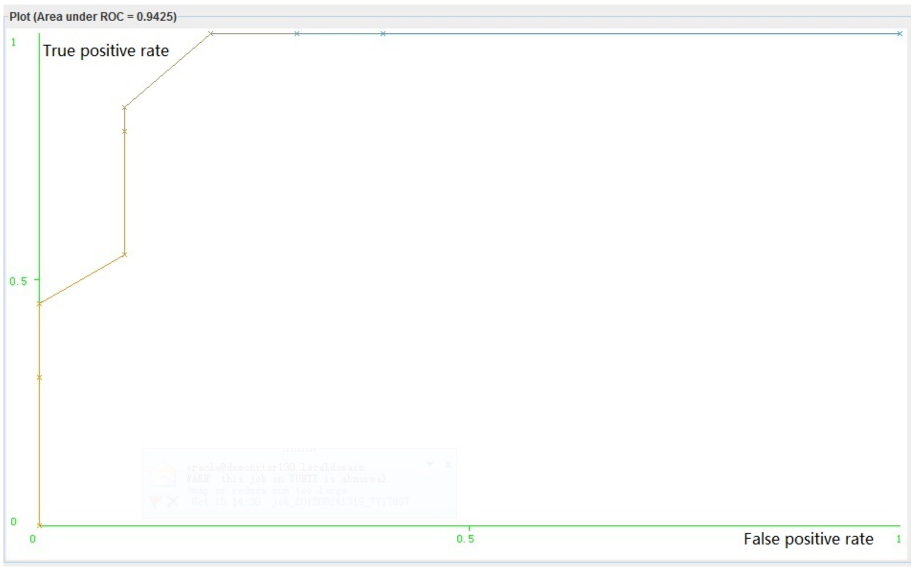


Figure 3.4: ROC curve of classifier

The final decision tree is shown in Figure C.1 in Appendix C. The root of the tree is the feature “degree_mean”, if $degree\_mean > 31$ which means that the average number of connections for each developer is more than 31, the project is classified as “problematic”.

The weights for different features in logistic regression model are shown in Table 3.7. Generally speaking, negative value means this feature is in favor of the “healthiness” of the project while the positive value means this feature goes against the “healthiness” of the project. Therefore, for the positive features, we hope the value of this feature is small and for the negative features, we hope the value grows bigger for the “healthiness” of the project. For example, we expect the value of “cluster_coefficient_mean” grows bigger to let the network turn “healthier”.

We are still interested in two important questions: 1) how does a single feature contribute to the final model performance? 2) whether can we improve the current model performance.

To answer question (1), we remove “cluster_coefficient” related features from the feature set. Then we compare the model performance with the model which contains all features. We can see from the results, the adding of feature “cluster_coefficient” improves the prediction model. More importantly, this observation indicates that it is possible to improve the prediction model again by adding more good features. This will be our future work.

Variable	weight
degree_centrality	239.3423
degree_mean	−6.5934
degree_variance	53.5615
betweenness_mean	−9.3657
betweenness_variance	8.8004
normalized_closeness	2119.6052
closeness_mean	24.6526
closeness_variance	−1829.3567
assortativity	17.3721
diameter	1.334
normalized_diameter	−13.742
cluster_1_4	0.2429
cluster_2_4	24.6521
cluster_3_4	−13.8257
cluster_1_4_variance	1585.8335
cluster_2_4_variance	−212.6712
cluster_3_4_variance	−1339.3603
density	−13.1869
eigen$_mean$	886.5269
cluster_coefficient_mean	−102.9087
cluster_coefficient_variance	235.3408
edge_delta	−12.6035
Intercept	−143.2342

Table 3.7: Feature weight for logistic regression

We experiment feature removing on *RandomForest* model since it achieves the best result so far. The results of including and excluding feature *CC* are shown in Table 3.8. We conclude that feature *CC* contributes significantly to our model performance, at least for *RandomForest* model.

	Precision	**Recall**	**F-Measure**	**AUC**
Excluding *CC*	0.809	0.72	0.671	0.625
Including *CC*	0.785	**0.755**	**0.745**	**0.942**

Table 3.8: Model performance comparison before adding feature *CC* and after adding it

CHAPTER 4

Recommendations in Developer Social Network

In the previous chapters, we have seen different developer social network structures and how these different structures indicate or result in the final success or failure of the project. We model the developer social network and give "healthy" scores for the constructed network to let the project managers know whether current social networks among its developers is "healthy" or "problematic". More importantly, project managers also want to know how to optimize the current developer social network. This is useful for project managers to make decisions in advance to avoid project failure due to ill-formed developer social networks. In this chapter, we clarify the recommendation problem in developer social networks and implement two social recommendation algorithms to generate recommendations with the aim of optimizing the current social network among the project developers.

4.1 Social Recommendation in Developer Social Networks

Recommender systems suggest items, information or people from the analysis of profiles of the current user and other users who have a similar preference [17]. With the development of Internet and e-commerce, recommender systems become more and more important to improving the efficiency of information retrieval (IR) with the ever-increasing volume of information available. In the present research, we consider recommendation of similar developers to the current developers which is also referred to as link prediction problem in social networks [28]. Given the current connections among the developers, we want to know which relationships are most needed to be added to the current network, i.e., we choose unconnected developer pairs that are most likely to optimize the network if added to recommend. Charu et al. [2] gave a review of this field (Chapter 9).

The link prediction problem can be formalized as follows: Given a user pair

(u, v), estimate the probability of creation of the link $u \rightarrow v$ and is often solved by approaches classified into the following three categories:

1. **Matrix completion**: treats link prediction as a matrix completion problem [33]. Given a partial known adjacency matrix representing users' relationships, matrix completion methods predict unknown elements in the matrix based on which recommendations are generated.
2. **Random walk**: explore network structure using Markov transition to infer transition probability between any two given vertices in the graph [4].
3. **Latent class models**: models the vertices in the social network and gives latent class labels to the vertices. Recommendations are made based on the distance measured in these latent subspaces [3].

In the following sections, we apply two existing social recommendation algorithms to give recommendations about connections between developers and give detailed experimental results on our real-world dataset.

4.2 Random Walks and Random Walk with Restarts

Random walk algorithms refer to a bunch of iterative graph mining algorithms which transit (random walks) from one vertex to other neighboring vertex according to the transitory probability between these two vertices until the statuses of any vertices do not change (convergence). Through random walk after enough iterations, the predefined statuses of the vertices will be stable. This stable probability can be thus used for global (impersonalized) recommendation. If a certain bias (a probability) is given to a vertex, then the next step will stay in the vertex with this bias probability. Finally the stable probability will be the measure of closeness of other vertices and the current vertex. This extension is called Random Walk with Restarts (RWR) [27]. Recommendation based on RWR considers both the network structure and vertex bias, therefore the recommended results are personalized.

Our RWR method follows the method proposed by [27] which is defined as follows:

Transition probabilities of jumping from any node to an adjacent node:

$$P(node(t+1) = j | node(t) = i) = a_{ij}/a_i = p_{ij}$$

where $a_i = \sum_{j=1}^{n} a_{ij}$.

Adjacency matrix A represents the similarity degree (or weight of relationship) between any two nodes.

$$A(i,j) = weight\ on\ edge\ from\ node\ i\ to\ node\ j$$

Transition matrix P is defined based on the Adjacency matrix A as follows:

$$P(i,j) = A(i,j)/\sum_i A(i,j)$$

Take the network in Figure 2.1 for example, its transition matrix is shown as follows:

$$\begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0.5 & 0.5 & 0 \end{pmatrix}$$

$$P = \begin{pmatrix} 0 & 1/2 & 0 & 1/2 \\ 1/3 & 0 & 1/3 & 1/3 \\ 0 & 1 & 0 & 0 \\ 1/2 & 1/2 & 0 & 0 \end{pmatrix}$$

We explain the concepts of Random Walk and Random Walk with Restart here. Starting from a specific vertex x, a Random Walk is performed repeatedly by choosing an edge from all the edges started from vertex x to another vertex according to the transition probability between the two vertices at each step. If with Restart, in every step there is a predefined constant probability α (which we denote as Restart ratio) to restart at vertex x in the next step. $s(t)$ is our target vector where its element $s(t)_i$ denotes the probability that the random walk stays at vertex i at step t. Let P be the transition matrix of the graph following the definition above "where its element $P_{i,j}$ denotes the conditional probability of vertex j being the next active vertex if the current random walk is at vertex i" [27]. Vector q is defined as an all-zero vector except for the element corresponding to the starting vertex x which is set to be 1, i.e. $q_x = 1$. Stationary probabilities for each vertex are obtained by calling Equation 4.1 iteratively until convergence is achieved (the change from the last iteration is less than a predefined threshold):

$$\mathbf{s}^{t+1} = (1-\alpha)P\mathbf{s}^t + \alpha\mathbf{q}. \tag{4.1}$$

Equation 4.1 is performed independently n times for n different vertices in the graph independently. Therefore, we will get $n-1$ possibilities for corresponding relationships at the maximum for each round and $n * (n-1)/2$ relationships at the maximum altogether. Because all the candidate relationships have a probability giving by the stationary probability, these candidates are ranked globally in descend order. We choose the top K candidates to recommend.

Parameter $\alpha \in [0,1]$ is the random walk restarting ratio which controls the ratio of personalization in the final recommendation. If α is set to be 0, the recommendation results are universal for all the developers, while if set to 1, the recommendation results are totally personalized. The complete description of our RWR algorithm is shown in Algorithm 1.

Algorithm 1: RWR recommendation algorithm

```
Input: Transition matrix $P_{n,n}$, developer set $D$ where $|D| = n$,
       recommendation list size $K$
Output: Recommended connections list $\Omega$, $|\Omega| = K$
1  for $d_i \in D$ do
2      Initialize vector $\mathbf{s}^{(0)} = \mathbf{0}$;
3      Initialize vector $\mathbf{q}$ as $\mathbf{q}_j := \begin{cases} 1 & \text{(if the } i = j\text{)} \\ 0 & \text{otherwise} \end{cases}$;
4      while $diff > CONVERGENCE\_THRESHOLD$ do
5          $\mathbf{s}^{t+1} = (1-\alpha)P\mathbf{s}^t + \alpha\mathbf{q}$;
6          $diff = abs(|\mathbf{s}^{t+1}| - |\mathbf{s}^t|)$;
7      end
8      Add $\mathbf{s}$ to recommendation list;
9  end
10 Ranking the recommendation list in decreasing order into vector $\hat{R}$;
11 $\Omega = \hat{R}[0:K]$;
12 return $\Omega$;
```

4.3 Ant Colony Recommendation Algorithm

Recently, a novel recommendation inspired by Ant Colony Recommendation (ACR) Algorithm was proposed which is suitable for complex network recommendation, particularly suitable for recommendation in bipartite-graph [49]. We introduce and implement ACR in this section and make comparisons with RWR algorithm for our task. ACR algorithm is a recommendation algorithm which is inspired by the well-known ant colony algorithm. It allocates a unique type pheromone to each of the foods (code files in our settings) or a group of foods, the ants (developers in our settings) exchange their pheromone with the food's when they eat the food (edit the file). The amount of the pheromone transmission is multiplied with a spreading factor γ which controls transmission rate. Therefore, after some interchange iterations, foods which are edited by some same developers will carry similar pattern of pheromone set and the same is to the developers. The process of pheromone transmission between ants and foods is illustrated in Figure 4.1. The edges in the bipartite graph denote the eating of food (editing of files in our settings), they are learnt one by one following their original editing order.

Bipartite representation of Developer Social Networks

A project is composed of elements belonging to two groups: the developer set A and the code file set F which can be naturally represented by two groups of vertices in Bipartite graph, where one of the two vertex sets denotes the developers and the other set denotes code files. Links only exist between two nodes from different sets, in case that the user has edited the file. A sample graph is shown in Figure 4.1 which contains 6 developers and 3 files and 9 editing records (extended from [49]).

Pheromone initialization

For initialization, we allocate every single code file an unique pheromone type Ph_f with amount 1.0. The developer pheromone is set to be empty for each developer at the very beginning.

Pheromone transmission

When a developer a_i edits a file f_j, they update their pheromone set according to each others pheromone set. The developer change her pheromone by adding the amount of the file's pheromone multiplied with a coefficient $\gamma \in (0, 1)$ which controls the spreading rate of pheromone between developers and code files. For the code file, its pheromone set is updated similarly. The editions are processed sequentially in their temporal order shown in Figure 4.1 (the labels on the edge are their order). After just a few times of editions, the pheromone on both the developers and code files will mix sequentially and sufficiently which is desired for recommendation which often suffers from data sparsity problem [49].

Pheromone evaporation

Developers' interests may change over time. Follow the pheromone evaporation mechanism designed in [49], we can capture the evolution of developers' interests by pheromone evaporation on both developers and on code files. Before computing the pheromone exchange between the developer and the code file, existing pheromone types on both the developers and files will evaporate in a rate calculated from the ratio of their amounts divided by the highest amount among

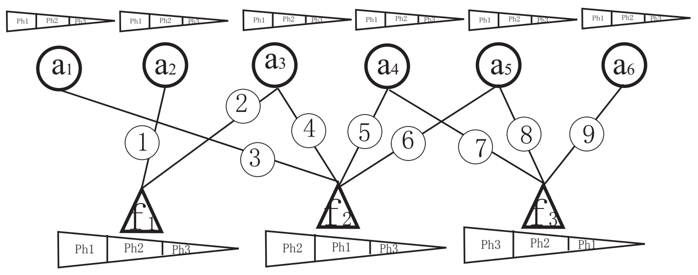


Figure 4.1: The bipartite graph of a software project with 6 developers and 3 files, the connections are edits with timestamps

all the existing pheromone types on the developer or code file (Equation 4.2).

$$Ph_{f_j} \times \exp\left(\frac{amount_{f_{j,k}} + \lambda}{Max_{k\in K}(amount_{a_{i,k}}) + \lambda} - 1\right) \tag{4.2}$$

where λ is the damping factor which is a constant and controls the speed of evaporation. λ is the only parameter that needs to be fine tuned about which we will discuss in the experiment section.

The complete pheromone update formulae for the ant and the food can therefore be described in Equation (4.3) and Equation (4.4) which follow the equations defined in [49] with slight modifications.

$$\mathbf{Ph}_{\mathbf{f_j}}^{\mathbf{(t+1)}} = \mathbf{Ph}_{\mathbf{f_j}}^{\mathbf{(t)}} \times \exp\left(\frac{amount_{f_{j,k}} + \lambda}{Max_{k\in K}(amount_{f_{j,k}}) + \lambda} - 1\right) + \gamma_{a_i} \times \mathbf{Ph}_{\mathbf{a_i}}^{\mathbf{(t)}} \tag{4.3}$$

$$\mathbf{Ph}_{\mathbf{a_i}}^{\mathbf{(t+1)}} = \mathbf{Ph}_{\mathbf{a_i}}^{\mathbf{(t)}} \times \exp\left(\frac{amount_{f_{j,k}} + \lambda}{Max_{k\in K}(amount_{a_{i,k}}) + \lambda} - 1\right) + \gamma_{f_j} \times \mathbf{Ph}_{\mathbf{f_j}}^{\mathbf{(t)}} \tag{4.4}$$

where γ_{a_i} is defined as the spreading rate of pheromone from developer i and γ_{f_j} is defined as the spreading rate from code file j. In practice, we set $\gamma_{a_i} = \frac{2}{|Ph_{a_i}|+1}$ and $\gamma_{f_j} = \frac{2}{|Ph_{f_j}|+1}$ which are decreasing functions of number of types of pheromone set already attached on the ant the food, respectively. Therefore, the more is the number of types of pheromone attached on the developer or file, the slower it spreads any kind of pheromone because this type of pheromone is less important for this developer or file compared with the ones with less number of pheromone types.

Similarity computation

The similarity between any two developers is defined:

$$Sim_{i,j} := \sum_{p \in (Pheromone_i \cap Pheromone_j)} amount_{i,p} * amount_{j,p}$$

Recommendation based on similarity The recommendation is relatively trivial by sorting the similarities in reversed order and recommends the top K relationships.

Altogether, the ACR algorithm is formally described below (with slight modifications from [49]):

Algorithm 2: Training phase of ACR

```
Input: $\mathcal{E}$, $\mathcal{V}$, $\mathcal{F}$
Output: Updated developer pheromone and code file pheromone
//Initialization
for $a \in \mathcal{V}$ do
    $Ph_a^{(0)} = \{\}$;
end
for $v_j \in F$ do
    $Ph_f^{(0)} = \{f : 1.0\}$;
end
//Training
for $edit_{af} \in \mathcal{E}$ do
    // Update developer pheromones
    for $Pheromone \in \mathbf{Ph_a^{(t)}}$ do
        // Evaporation and then transmission
        $Pheromone = Phermone \times \exp(\frac{amount_{Pheromone}+\lambda}{Max_{Ph_a}+\lambda} - 1) + \gamma_f \times Ph_f^{(t)}$
        // Cut off, $\sigma$ is a predefined threshold
        if $abs(Phermone) < \sigma$ then
            $Phermone = 0$;
        end
        $\mathbf{Ph_a^{(t+1)}} \leftarrow Phermone$;
    end
    // Update code file pheromones
    for $Pheromone \in \mathbf{Ph_f^{(t)}}$ do
        // Evaporation and Transmission
        $Pheromone = Phermone \times \exp(\frac{amount_{Pheromone}+\lambda}{Max_{Ph_f}+\lambda} - 1) + \gamma_a \times Ph_a^{(t)}$
        // Cut off
        if $abs(Phermone) < \sigma$ then
            $Phermone = 0$;
        end
        $\mathbf{Ph_f^{(t+1)}} \leftarrow Phermone$;
    end
end
```

4.4 Experiments and Results

We have implemented the previous algorithms on our developer social networks and did many experiments on real-world datasets to validate and compare the methods proposed above.

4.4.1 Data Set and Evaluation Metrics

Data Set

The projects for recommendation experiments are crawled the same way as in Section 3.1.1. The only difference is that the manually labeling phase is not needed for these projects because the model we learnt from the last chapter can give trustable predictions about the project as we have seen in the experimental section in the last chapter. For experimentation, we choose 15 projects as listed in Table B.1 in Appendix B.

Evaluation Metrics

To evaluate the effectiveness of recommendation results, however, is not easy and straightforward in our setting. Intuitively, we want to know two things: 1) Whether our recommendations are likely accepted by our developers; 2) Will the recommendations actually improve the social network?

Ranking AUC

Area Under the ROC Curve (*AUC*) is a measure widely used in ranking and recommendation tasks and is the de facto performance measure for link prediction task [33]. It gives measurement about appropriateness of the recommendation results. We capture a snapshot at the 3/4 time point of a project's lifecycle and generate recommendations based on this snapshot of the DSN. Then we check how many recommendations actually appear in the updated network after the rest 1/4 of the lifecycle.

$$AUC = \frac{\sum_{k=1}^{K}((\sum_{i=1}^{k} hit_i + \sum_{i=1}^{k-1} hit_i) * no\_hit_k)}{2 * \sum_{k=1}^{K} hit_k * \sum_{k=1}^{K} no\_hit_k} \tag{4.5}$$

where $hit_i := \begin{cases} 1 & \text{(if the } i th \text{ recommendation is matched)} \\ 0 & \text{(if the } i th \text{ recommendation is not matched)} \end{cases}$ and $no\_hit_i :=$ $\begin{cases} 0 & \text{(if the } i th \text{ recommendation is matched)} \\ 1 & \text{(if the } i th \text{ recommendation is not matched)} \end{cases}$ and K is the number of recommendations.

Model Improvement

There is an obvious shortcomings for AUC metric calculated on the match of recommendation and real connections. That is good recommendations do not necessarily appear in the real network afterwards. In fact, the calculation may skip some good recommendations and thus underestimate the real precision of the recommendations. On the other hand, the objective of our recommendation is to improve the developer social network.

Moreover, if the recommendation list include connections that the users need to or likely to connect, thus if these recommendations are accepted, the developer social network should be improved. Therefore, we are wondering whether the recommendations can really improve the social network so that the project is more "healthy" and the development efficiency is boosted. We can assume the recommendations are accepted by the developers so that the connections are all added to the current network. We are interested in the change in the score given by the model we have learnt in the last chapter.

$$Score\_Improve = predict\_score_{new} - predict\_score_{old}. \tag{4.6}$$

The bigger the $Score\_Improve$ is, the more obvious the improvement is, so that the better the recommendation.

4.4.2 Experiment Results

General results

We experimented **RWR** and **ACR** methods on our given project list. The results of **AUC** measure and **Score_Improve** measure are reported in Table 4.1 as follows. We also tested for different K values, i.e. the recommendation list size, the difference of performances of both methods.

Parameter tuning Restart rate α is an important parameter in RWR algorithm which controls personalization factor in recommendation results. We experiment different α's influence in the recommendation results. In general, the recommendation results are the best when $\alpha = 0.5$. The results are shown in Figure 4.2.

Similarly, damping factor λ controls the rate of pheromone evaporation on ants and foods. It is an important parameter that needs to be tuned carefully. The comparison results are shown in Figure 4.3. The higher is the value of λ

Table 4.1: Rating Results Evaluation

Algorithm		AUC	Score_Improve
RWR	$K = 5$	0.958	0.012
	$K = 10$	0.910	0.013
	$K = 15$	0.874	0.000
ACR	$K = 5$	0.667	0.000
	$K = 10$	0.533	0.011
	$K = 15$	0.530	0.002

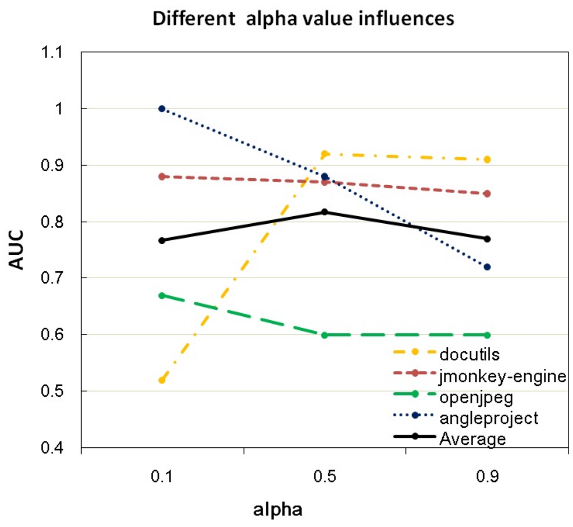


Figure 4.2: Impact of random walk restart parameter α on the recommendation results for RWR

means the pheromone evaporates slower. For our experiments, recommendation results are best when $\lambda = 50$.

Visualization of recommendations

The recommendation results are visualized using the Jung library[1]. The project manager can examine the recommendation results directly and make his/her own decisions based on the recommendations given automatically. Take project "ProtoBuf", an open source project initialed by Google, for example. Figure 4.4 is the recommendation results generated by RWR algorithm and Fig-

[1]http://jung.sourceforge.net/

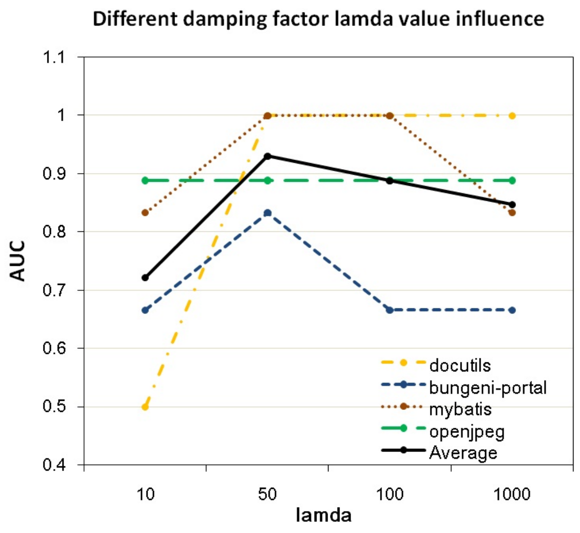


Figure 4.3: Impact of damping factor parameter λ on the recommendation results for ACR

ure 4.5 is the recommendation results generated by ACR algorithm. The red lines are the connections that are recommended. From our judgement, the recommendation results given by RWR are slightly better than those given by ACR.

4.5 Discussions

Social network recommendations especially link prediction is a relatively new research topic. Although developer social network recommendation can be cast as a special case of social network recommendation, it has its special properties that makes the problem unique and more interesting. For example, the relationship between developers is not a direct relationship like in Facebook. The relationships between developers are excavated through co-authorship of the same code file. More importantly, the purpose of recommendation in developer network is to optimize the network itself rather than recommending personal relationships. These differences impose challenges for developer social network recommendation.

Another observation that is obvious through visualization is that the recommendation made by ACR is symmetrical while it is asymmetric for RWR method. However, ACR cannot recommend for developers who don't have any existing connections with other developers. This phenomenon is called "cold-start" in recommendation literature. This is a big challenge for social network

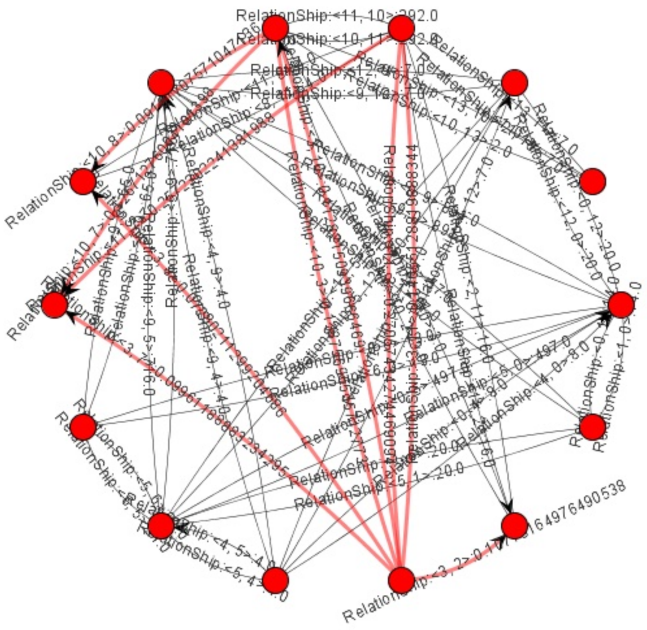
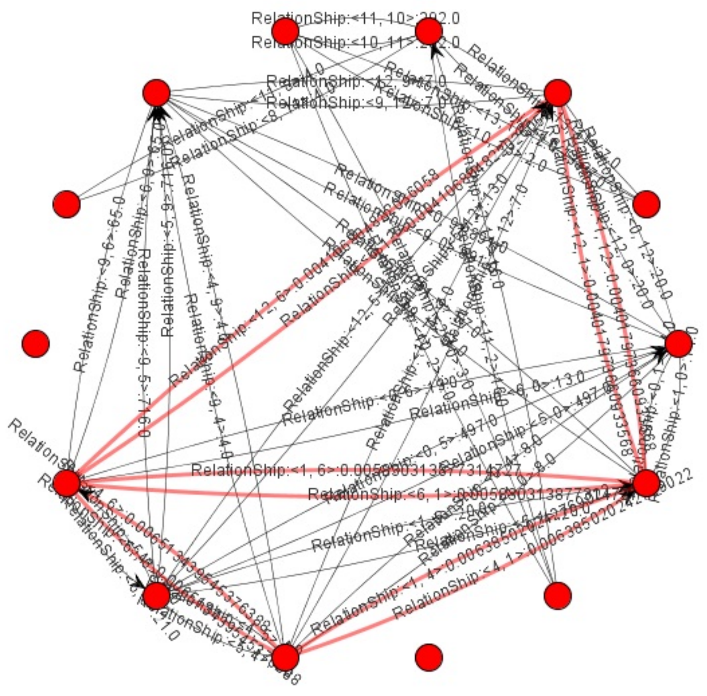


Figure 4.4: Visualization of recommendation results for Random Walk with Restarts (Red lines are recommendations)

recommendation, too.

As we have pointed out, developer recommendation has its own properties and deserves further researches in various aspects. Here we would like to point out some future directions of further research. First, content information such as affiliation information and edit content can be employed. Second, the network is best be modeled as a bipartite-graph with two groups of vertices representing developers and code files just as our second method did. How to recommend in these bipartite-graphs is a new problem in social recommendation context. Last but not least, we only concern about developer recommendations in this work, actually, other types of recommendations such as code file recommendations and project recommendations are also needed for project development process improvement.



RelationShip:<11, 10>:202.0
RelationShip:<10, 11>:202.0
RelationShip:<9, 6>:65.0
RelationShip:<0, 5>:71.0
RelationShip:<5, 0>:71.0
RelationShip:<4, 9>:4.0
RelationShip:<9, 4>:4.0
RelationShip:<0, 12>:20.0
RelationShip:<12, 0>:20.0
RelationShip:<1, 6>:0.005880313877314727
RelationShip:<6, 1>:0.005880313877314727
RelationShip:<0, 5>:497.0
RelationShip:<5, 0>:497.0
RelationShip:<6, 0>:13.0

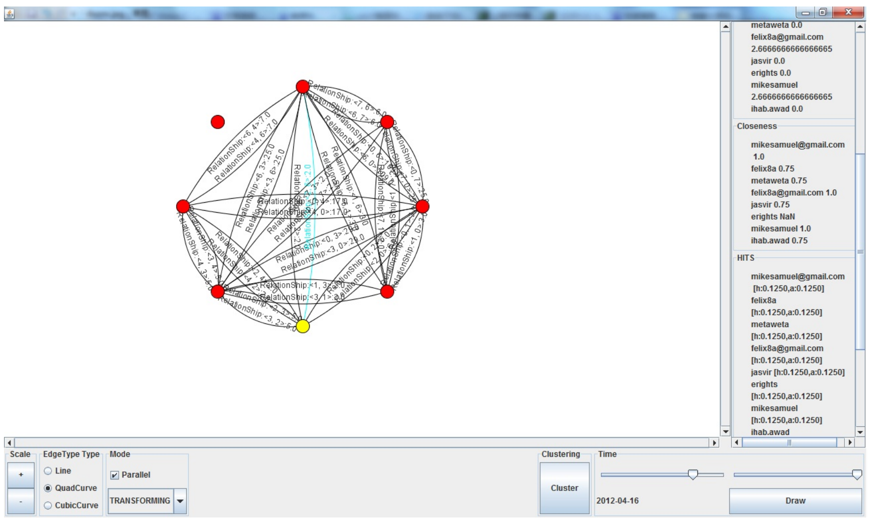
Figure 4.5: Visualization of recommendation results for ACR recommendation (Red lines are recommendations)

CHAPTER 5

Developer Social Network Optimization System (DSNOS)

Based on the theoretical works above, a Developer Social Network Optimization System is proposed and implemented, aiming to provide "visualization, analysis and recommendation" functionalities for optimization the social network of software developers. To the best of our knowledge, it is the first of its kind and we would like to describe it in more detail with the hope of providing some help for the analysis and optimizing of ongoing software projects.

The main Graphic User Interface (GUI) of DSNOS (Figure 5.1) is illustrated in Figure 5.1. On the left panel, the DSN graph is visualized; Some of the important features of this network are listed on the right panel. In the bottom part of the GUI, we list some useful control buttons which trigger clustering, evolution dynamics, classification results and recommendation results view of the selected project.



metaweta 0.0
felix8a@gmail.com
2.6666666666666665
jasvir 0.0
erights 0.0
mikesamuel
2.6666666666666665
ihab.awad 0.0

Closeness

mikesamuel@gmail.com
1.0
felix8a 0.75
metaweta 0.75
felix8a@gmail.com 1.0
jasvir 0.75
erights NaN
mikesamuel 1.0
ihab.awad 0.75

HITS

mikesamuel@gmail.com
[h:0.1250,a:0.1250]
felix8a
[h:0.1250,a:0.1250]
metaweta
[h:0.1250,a:0.1250]
felix8a@gmail.com
[h:0.1250,a:0.1250]
jasvir [h:0.1250,a:0.1250]
erights
[h:0.1250,a:0.1250]
mikesamuel
[h:0.1250,a:0.1250]
ihab.awad

EdgeType Type: [✓ QuadCurve]
Mode: [✓ Parallel] TRANSFORMING
Time: 2012-04-16

Figure 5.1: The main GUI of DSNOS

5.1 Functionalities Highlights

DSNOS mainly provides the following functionalities:

- Project Crawling: get the source code and change history of the project automatically;
- Developer Social Network Construction: based on the data crawled from the project repository, developer social network is constructed and its features is analyzed and displayed using visualization module.
- Developer Social Network Visualization: the visualization of DSN, classification model and recommendation results.
- Developer Social Network Analysis: display the network features.
- Developer Social Network Healthiness Classification: predict the label of the given project social network using the trained model.
- Recommendation in Developer Social Network: recommend relationships and display them on the existing network.

Here we would like to describe some of the features of DSNOS in more detail.

5.1.1 Visualization

Visualization is an important means for social network analysis because it provides most direct knowledge about the structure of the network. Moreover, our classification results and recommendation results also need to be visualized so that project manager's own experiences and intuitions can be better utilized as assistances.

Temporal effects

Figure 5.2 are the screenshots of DSN taken from different timestamps. We could see the growth of DSN over time and development process.

Clustering

We also implemented a graphical interface to view clustering results for a given network. Figure 5.3 shows the clustering results when we remove different number of edges from the network. The gray ones are the removed edges and the black ones are those maintained.

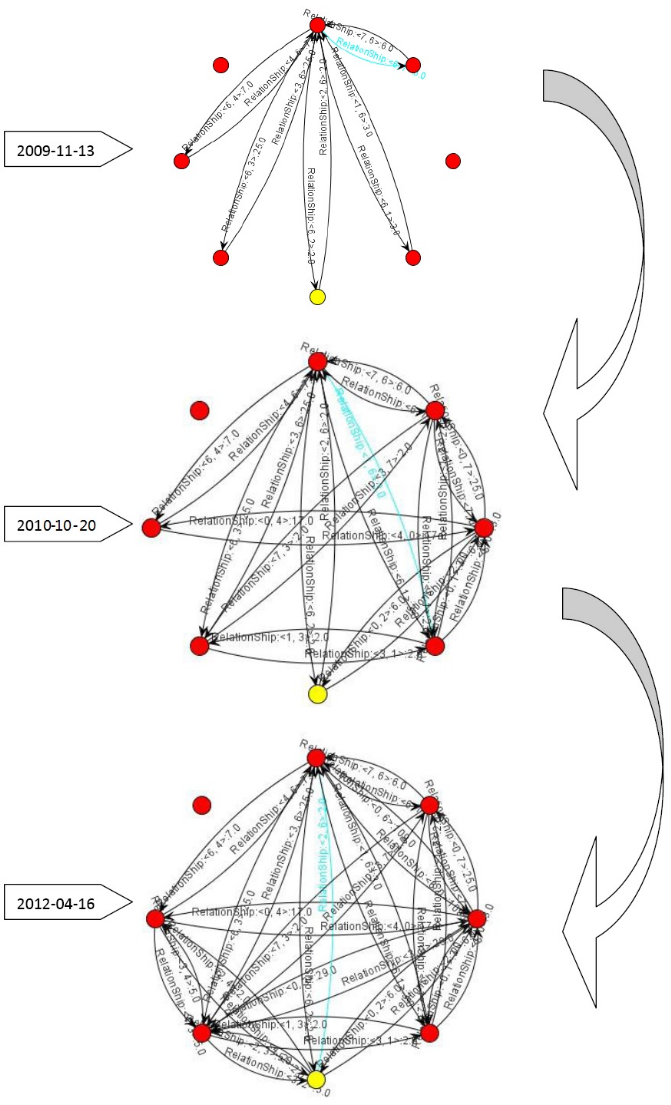


Figure 5.2: Visualization of DSN's change over time

Network features The panel on the right side lists the features extracted from the developer social network of the given project. Figure 3.3 are screeshots of the feature values of the project "GoogleTest".

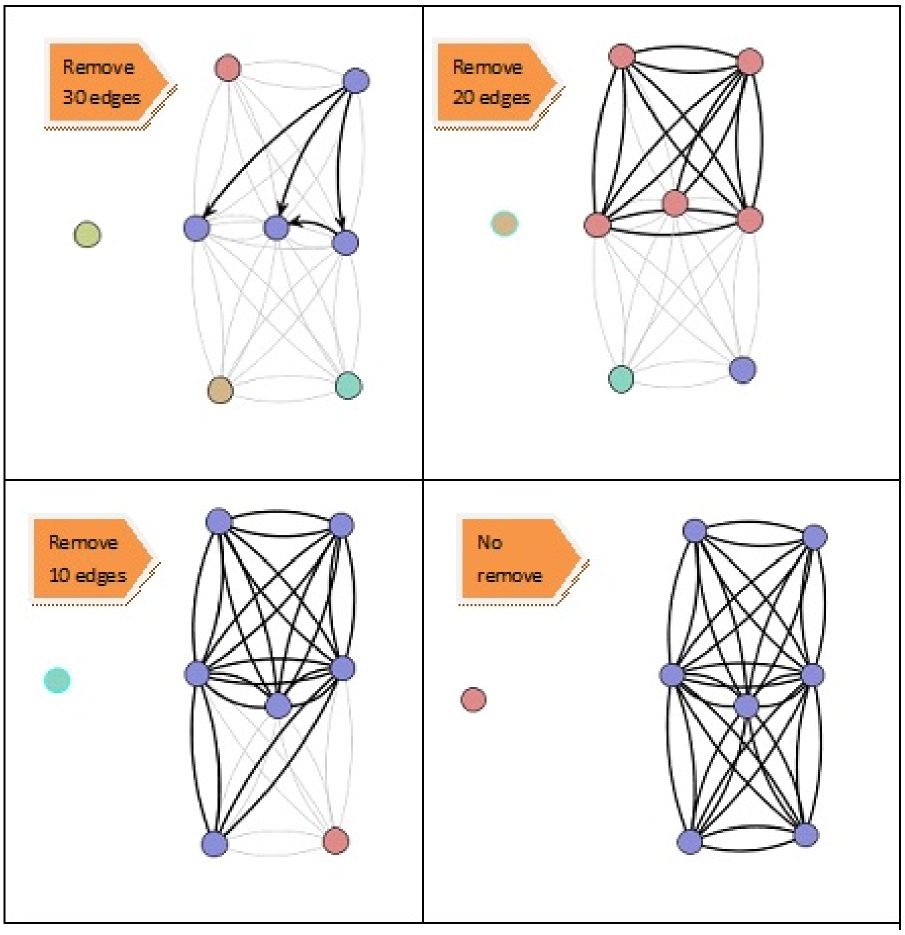


Figure 5.3: Illustration of clustering results

5.2 Implementation Detail and Application

The DSNOS system is implemented using Java. The main reason of choosing Java is that there are plentiful and powerful third party packages that we can depend on. The following third party utilities (open-source) are used to facilitate the implementation.

- Jung: Jung ("Java Universal Network/Graph Framework"), is an open source library that provides an extendible API based on Java for the representing, analysis, and visualization of graphic or network data. JUNG integrates lots of functions that provide graph/network visualization and attribute computation and analyzing. As an open-source library, JUNG also provides interfaces that can be easily extended using standard Java language to support more interactive and customized functionalities.

- Weka: We have introduced basic usage of Weka in Section 2.4. In DSNOS, Weka library is embedded and provide all the functionalities related to machine learning model training and evaluation. It provides evaluation functionalities together with all the machine learning methods we have implemented in Section 3.2.

5.3 Future Works

We realize DSNOS is the first of its kind. Although we try our best to provide most useful and interesting functionalities as completely as possible, there does remain many functional points that could be improved in the near future.

- Support more types of software repositories.
- Optimize the GUI of DSNOS to provide more considerate information.
- Validate its effectiveness in more real-world projects and collect feedbacks.

CHAPTER 6

Conclusions and Future Works

6.1 Conclusions and Discussions

In this dissertation, we focussed on the analysis and optimization of developer social networks in order to facilitate development process and increase software quality. We have observed that the structure of developer social networks from different projects differ greatly. Both researchers and practitioners believe that these differences would have a great impact on the success of project. The question is: what kind of networks are "healthy" networks and what kind of are "problematic" networks. This is a new, while very important, question to which we try to give our answer. We trained several machine learning models based on the features extracted from different kinds of developer social networks. These networks were labeled manually. The 10-fold cross validation results showed that our models have a high prediction precision on the unlabeled new projects which means the model can tell a "healthy" network from "problematic" network with a strong confidence, given the criterions described in Section 3.1.6. Moreover, for "problematic" networks, we also want to know how to optimize it. Social recommendation algorithms are proposed to recommend the most relative relationships to developers. Experiments showed our recommendations match with real relationships with a high probability and very possibly will optimize the developer social network in terms of its "healthiness".

About project labeling

In classification phase, the training data are labeled manually according to the criteria that we have consulted many previous works and empirical experiences. Although it is fairly reasonable, it is better to have a deeper investigation into the project itself, using techniques such as questionnaire and lifelong tracing which records important project indexes for the whole lifecycle of the project. Thus can we have more justification and confidence to label the project.

About recommendation

Recommendation in developer social network is a brand new problem in spite of its importance in optimizing developer social networks. There are at least two major challenges here: 1) The recommendation problem in DSN is more naturally solved by recommendation algorithms based on Bipartite graph which is more complex than common graph, as we have done in ACR; 2) The evaluation of recommendation results is not as easy as the evaluation of classification results especially without user feedbacks. We gave our solutions to these two challenges and the results are satisfactory. Still, we are not 100% percent sure about its effectiveness for project managers. It is important for collecting feedbacks from project managers.

6.2 Future Work

To better achieve the objectives proposed in this dissertation, there are still many future directions that may inspire more researchers.

Improve the classification and recommendation performances

To further improve the classification results, we can try different combinations of features and we need to obtain more training projects in order to let the model to be less overfitted to the training data.

For recommendation, both RWR and ACR algorithms consider developer (vertex) based recommendation. However, we realize there is a gap between vertex based edge (relationship) recommendation and graph based edge recommendation. Strictly speaking, graph based edge prediction should be more relevant to our objective because our optimization goal is to optimize the graph globally. This is a big challenge for us to consider graph based recommendation.

Evaluate DSNOS by introducing it to some ongoing projects

DSNOS system aims to provide service for real-world project. We need to validate its effectiveness through evaluation and feedbacks from industries.

Considering influence

Influence is the measurement of an individual's influence on other individuals or the network as a whole. It is important for software project analysis and improvement through finding the most influential developers in the developer network. There are already some research work concerning on how to calculate the vertex influences precisely and how to maximize the utility of existing influences. We plan to apply these researches to our developer social network analysis system.

Bibliography

[1] Adomavicius, G., and Tuzhilin, A. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE Trans. on Knowl. and Data Eng. 17*, 6 (June 2005), 734–749.

[2] Aggarwal, C. C., Ed. *Social Network Data Analytics.* Springer, 2011.

[3] Airoldi, E. M., Blei, D. M., Fienberg, S. E., and Xing, E. P. Mixed membership stochastic blockmodels. *J. Mach. Learn. Res. 9* (June 2008), 1981–2014.

[4] Backstrom, L., and Leskovec, J. Supervised random walks: predicting and recommending links in social networks. In *Proceedings of the fourth ACM international conference on Web search and data mining* (New York, NY, USA, 2011), WSDM '11, ACM, pp. 635–644.

[5] Barabasi, A., Jeong, H., Neda, Z., Ravasz, E., Schubert, A., and Vicsek, T. Evolution of the social network of scientific collaborations. *The Journal of Physica A 311*, 3-4 (2002), 590–614.

[6] Bird, C., Pattison, D., D'Souza, R., Filkov, V., and Devanbu, P. Latent social structure in open source projects. In *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering* (New York, NY, USA, 2008), SIGSOFT '08/FSE-16, ACM, p-p. 24–35.

[7] Bishop, C. M. *Pattern Recognition and Machine Learning (Information Science and Statistics).* Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[8] Bouktif, S., Gueheneuc, Y.-G., and Antoniol, G. Extracting change-patterns from cvs repositories. In *Proceedings of the 13th Working Conference on Reverse Engineering* (Washington, DC, USA, 2006), WCRE '06, IEEE Computer Society, pp. 221–230.

[9] Breiman, L. Random forests. *Mach. Learn. 45*, 1 (Oct. 2001), 5–32.

[10] Breiman, L., Friedman, J. H., Olshen, R. A., and Stone, C. J. *Classification and regression trees.* Wadsworth Publishing Company, 1984.

[11] Carmagnola, F., Vernero, F., and Grillo, P. Sonars: A social networks-based algorithm for social recommender systems. In *Proceedings of the 17th International Conference on User Modeling, Adaptation, and Personalization: formerly UM and AH* (Berlin, Heidelberg, 2009), UMAP '09, Springer-Verlag, pp. 223–234.

[12] Carrington, P. J., Scott, J., and Wasserman, S., Eds. *Models and Methods in Social Network Analysis.* Cambrige University Press, 2005.

[13] Clements, M., de Vries, A. P., and Reinders, M. J. T. Optimizing single term queries using a personalized markov random walk over the social graph. In *Workshop on Exploiting Semantic Annotations in Information Retrieval (ESAIR)* (Mar. 2008).

[14] Fouss, F., Pirotte, A., Renders, J.-M., and Saerens, M. Random-walk computation of similarities between nodes of a graph with application to collaborative recommendation. *IEEE Trans. on Knowl. and Data Eng. 19*, 3 (Mar. 2007), 355–369.

[15] Freeman, L. C. A set of measures of centrality based on betweenness. *Sociometry 40*, 1 (Mar. 1977), 35–41.

[16] Gao, Y., and Freeh, V. andMadey, G. Analysis and modeling of the open source software community. In *NASOS* (2003).

[17] Georg Groh, C. E. LaTeX : Recommendations in taste related domains: collaborative filtering vs. social filtering. In *GROUP07* (2007), pp. 473–480.

[18] Guo, L., Ma, Y., Cukic, B., and Singh, H. Robust prediction of fault-proneness by random forests. In *Proceedings of the 15th International Symposium on Software Reliability Engineering* (Washington, DC, USA, 2004), ISSRE '04, IEEE Computer Society, pp. 417–428.

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

[20] Herbsleb, J. D., and Mockus, A. An empirical study of speed and communication in globally distributed software development. *IEEE Transactions on Software Engineering 29* (2003), 481–494.

[21] Hosmer, D. W., and Lemeshow, S. *Applied logistic regression (Wiley Series in probability and statistics)*, 2 ed. Wiley-Interscience Publication, Sept. 2000.

[22] Hotho, A., Jäschke, R., Schmitz, C., and Stumme, G. Information retrieval in folksonomies: search and ranking. In *Proceedings of the 3rd European conference on The Semantic Web: research and applications* (Berlin, Heidelberg, 2006), ESWC'06, Springer-Verlag, pp. 411–426.

[23] Ichii, M., Hayase, Y., Yokomori, R., Yamamoto, T., and Inoue, K. Software component recommendation using collaborative filtering. In *Search-Driven Development-Users, Infrastructure, Tools and Evaluation, 2009. SUITE '09. ICSE Workshop on* (2009), pp. 17–20.

[24] Kagdi, H., Collard, M. L., and Maletic, J. I. A survey and taxonomy of approaches for mining software repositories in the context of software evolution. *J. Softw. Maint. Evol. 19*, 2 (Mar. 2007), 77–131.

[25] Kang, U., Tsourakakis, C. E., and Faloutsos, C. Pegasus: A petascale graph mining system. *Data Mining, IEEE International Conference on 0* (2009), 229–238.

[26] Karamon, J., Matsuo, Y., and Ishizuka, M. Generating useful network-based features for analyzing social networks. In *Proceedings of the 23rd national conference on Artificial intelligence - Volume 2* (2008), AAAI'08, AAAI Press, pp. 1162–1168.

[27] Konstas I., Stathopoulos V., J. J. On social networks and collaborative recommendation. In *32nd International ACM SIGIR conference on Research and development in information retrieval SIGIR 09* (2009), pp. 195–202.

[28] Liben-Nowell, D., and Kleinberg, J. The link prediction problem for social networks. In *Proceedings of the twelfth international conference on Information and knowledge management* (New York, NY, USA, 2003), CIKM '03, ACM, pp. 556–559.

[29] Liu, L., Tang, J., Han, J., and Yang, S. Learning influence from heterogeneous social networks. *Data Mining and Knowledge Discovery* (Mar. 2012), 1–34.

[30] Lopez-Fernandez, L., Robles, G., and Gonzalez-Barahona, J. M. Applying social network analysis to the information in cvs repositories. In *International Workshop on Mining Software Repositories* (2004), pp. 101–105.

[31] Madadhain, J., Fisher, D., Smyth, P., White, S., and Boey, Y. Analysis and visualization of network data using jung. *Journal of Statistical Software 10* (2005), 1–35.

[32] Martinez-Romo, Robles, G., Gonzalez-Barahona, J. M., and Ortuno-Perez, M. Using social network analysis techniques to study collaboration between a floss community and a company. In *OSS'08* (2008), pp. 171–186.

[33] Menon, A. K., and Elkan, C. Link prediction via matrix factorization. In *Proceedings of the 2011 European conference on Machine learning and knowledge discovery in databases - Volume Part II* (Berlin, Heidelberg, 2011), ECML PKDD'11, Springer-Verlag, pp. 437–452.

[34] Menzies, T., Stefano, J. S. D., and Cunanan, C. Mining repositories to assist in project planning and resource allocation. In *International Workshop on Mining Software Repositories* (2004).

[35] Mierle, K. B., Mierle, K. B., Laven, K., Laven, K., Roweis, S. T., Roweis, S. T., Wilson, G. V., and Wilson, G. V. Cvs data extraction and analysis: A case study. Tech. rep., University of Toronto, 2004.

[36] Nagappan, N., Ball, T., and Zeller, A. Mining metrics to predict component failures. In *Proceedings of the 28th international conference on Software engineering* (New York, NY, USA, 2006), ICSE '06, ACM, pp. 452–461.

[37] Newman, M. E. J. Mixing patterns in networks. *Physical Review E 67*, 2 (2003), 026126.

[38] Nooy, W., Mrvar, A., and Batagelj, V. *Exploratory social network analysis with pajek.* Cambridge University Press, 2005.

[39] Norick, B., Krohn, J., Howard, E., Welna, B., and Izurieta, C. Effects of the number of developers on code quality in open source software: a case study. In *Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement* (New York, NY, USA, 2010), ESEM '10, ACM, pp. 62:1–62:1.

[40] Robles-Martínez, G., González-Barahona, J. M., Centeno-González, J., Matellán-Olivera, V., and Rodero-Merino, L. Studying the evolution of libre software projects using publicly available data. In *Proceedings of the ICSE 3rd Workshop on Open Source* (2003).

[41] Safavian, S. R., and Landgrebe, D. A survey of decision tree classifier methodology. *Systems, Man and Cybernetics, IEEE Transactions on 21*, 3 (1991), 660–674.

[42] Sebastiani, F. Machine learning in automated text categorization. *ACM Comput. Surv. 34*, 1 (Mar. 2002), 1–47.

[43] Snijders, T. A. B. The statistical evaluation of social network dynamics. *Sociological Methodology 31*, 1 (Jan. 2001), 361–395.

[44] Storey, M.-A. D., Čubranić, D., and German, D. M. On the use of visualization to support awareness of human activities in software development: a survey and a framework. In *Proceedings of the 2005 ACM symposium on Software visualization* (New York, NY, USA, 2005), SoftVis '05, ACM, pp. 193–202.

[45] Tang, J., Zhang, J., Yao, L., Li, J., Zhang, L., and Su, Z. Arnetminer: extraction and mining of academic social networks. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining* (New York, NY, USA, 2008), KDD '08, ACM, pp. 990–998.

[46] Tong, H., Faloutsos, C., and Pan, J.-Y. Fast random walk with restart and its applications. *Data Mining, IEEE International Conference on 0* (2006), 613–622.

[47] Wagstrom, P., Herbsleb, J., and Carley, K. A Social Network Approach To Free/Open Source Software Simulation. *Proceedings of the 1st International Conference on Open Source Systems, Genova, 11th-15th July* (2005).

[48] Walsh, B. Markov chain monte carlo and gibbs sampling. http://nitro.biosci.arizona.edu/courses/EEB596/handouts/Gibbs.pdf, 2004.

[49] Wang, Y., Liao, X., Wu, H., and Wu, J. Incremental collaborative filtering considering temporal effects. http://arxiv.org/pdf/1203.5415.pdf, 2012.

[50] Weissgerber, P., Pohl, M., and Burch, M. Visual data mining in software archives to detect how developers work together. In *Proceedings of the Fourth International Workshop on Mining Software Repositories* (Washington, DC, USA, 2007), MSR '07, IEEE Computer Society, pp. 9–.

[51] Witten, I. H., Frank, E., and Hall, M. A. *Data Mining: Practical Machine Learning Tools and Techniques*, 3. ed. Morgan Kaufmann, Amsterdam, 2011.

[52] Xie, X., Poshyvanyk, D., and Marcus, A. Visualization of cvs repository information. In *WCRE '06: Proceedings of the 13th Working Conference on Reverse Engineering* (Washington, DC, USA, 2006), IEEE Computer Society, pp. 231–242.

[53] Xu, J. *Mining and modeling the open source software community.* PhD thesis, Notre Dame, IN, USA, 2007. AAI3299241.

[54] Xu, J., Gao, Y., Christley, S., and Madey, G. A topological analysis of the open source software development community. In *Proceedings of the Proceedings of the 38th Annual Hawaii International Conference on System Sciences - Volume 07* (Washington, DC, USA, 2005), HICSS '05, IEEE Computer Society, pp. 198.1–.

[55] Ye, Y., Nakakoji, K., and Yamamoto, Y. Reducing the cost of communication and coordination in distributed software development. In *Proceedings of the 1st international conference on Software engineering approaches for offshore and outsourced development* (Berlin, Heidelberg, 2007), SEAFOOD'07, Springer-Verlag, pp. 152–169.

[56] Yildirim, H., and Krishnamoorthy, M. S. A random walk method for alleviating the sparsity problem in collaborative filtering. In *Proceedings of the 2008 ACM conference on Recommender systems* (New York, NY, USA, 2008), RecSys '08, ACM, pp. 131–138.

[57] Yu, L., and Ramaswamy, S. Mining cvs repositories to understand open-source project developer roles. In *Proceedings of the Fourth International Workshop on Mining Software Repositories* (Washington, DC, USA, 2007), MSR '07, IEEE Computer Society, pp. 8–.

[58] Yu, S. J. The dynamic competitive recommendation algorithm in social network services. *Inf. Sci. 187* (Mar. 2012), 1–14.

[59] Zimmermann, T. Preprocessing cvs data for fine-grained analysis. In *International Workshop on Mining Software Repositories* (2004), pp. 2–6.

[60] Zimmermann, T., and Nagappan, N. Predicting defects using network analysis on dependency graphs. In *Proceedings of the 30th international conference on Software engineering* (New York, NY, USA, 2008), ICSE '08, ACM, pp. 531–540.

APPENDIX A

Classification Project List

The detail of 30 projects for model training are list in Table A.1 below.

Project name	Repository	Status
twitculr	http://code.google.com/p/twitcurl/	Fine
GoogleTest	http://code.google.com/p/googletest/	Fine
FileZilla	http://sourceforge.net/projects/filezilla/	Fine
hbase	http://hbase.apache.org/	Problematic
tomcat	http://tomcat.apache.org/	Fine
struts2	http://struts.apache.org/2.x/	Fine
maven-3	http://maven.apache.org/	Problematic
gwt-google-apis	http://code.google.com/p/gwt-google-apis/	Fine
openmeetings	http://code.google.com/p/openmeetings/	Fine
mybatis	http://code.google.com/p/mybatis/	Fine
huggle	http://code.google.com/p/huggle/	Fine
heidisql	http://code.google.com/p/heidisql/	Fine
ganglia	http://ganglia.sourceforge.net/	Fine
corsix	http://code.google.com/p/corsix-th/	Fine
chrometophone	http://code.google.com/p/chrometophone/	Problematic
googleappengine	http://code.google.com/p/googleappengine/	Fine
chromium	http://code.google.com/p/chromium/	Problematic
protobuf	http://code.google.com/p/protobuf/	Fine
google-caja	http://code.google.com/p/google-caja/	Fine
v8	http://code.google.com/p/v8/	Problematic
flylinkdc	http://code.google.com/p/flylinkdc/	Fine
DocBook	http://sourceforge.net/projects/docbook/	Problematic
Docutils	http://docutils.sourceforge.net/	Fine
mamp	http://sourceforge.net/projects/mamp/	Fine
angleproject	http://code.google.com/p/angleproject/	Fine
gdata	http://code.google.com/p/gdata/	Problematic
flexigrid	http://code.google.com/p/flexigrid/	Fine
boost	http://www.boost.org/	Problematic
tcl	http://tcl.sourceforge.net/	Problematic
openjpeg	http://code.google.com/p/openjpeg/	Fine

Table A.1: Training project list

APPENDIX B

Recommendation Project List

The detail of 15 projects for recommendation experiments are list in Table B.1 below.

Project name	Repository
krank	http://krank.googlecode.com/svn/trunk/
jmonkeyengine	http://jmonkeyengine.googlecode.com/svn/trunk/
zxing	http://zxing.googlecode.com/svn/trunk/
jpcsp	http://jpcsp.googlecode.com/svn/trunk/
slim3	http://slim3.googlecode.com/svn/trunk/
mobicents	http://mobicents.googlecode.com/svn/trunk/
gwt-google-apis	http://gwt-google-apis.googlecode.com/svn/trunk/
allforgood	http://allforgood.googlecode.com/svn/trunk/
googletransitdatafeed	http://xbmc-addons-chinese.googlecode.com/svn/trunk/
fb2pdf	http://fb2pdf.googlecode.com/svn/trunk/
bungeni-portal	http://bungeni-portal.googlecode.com/svn/bungeni.main/trunk/
native_client	http://src.chromium.org/native$_c$*lient/trunk/src/native_client*
miranda	http://miranda.googlecode.com/svn/trunk/
libphonenumber	http://libphonenumber.googlecode.com/svn/trunk/
mdsp	http://mdsp.googlecode.com/svn/trunk/

Table B.1: Recommendation project list

APPENDIX C

Classification Model Examination

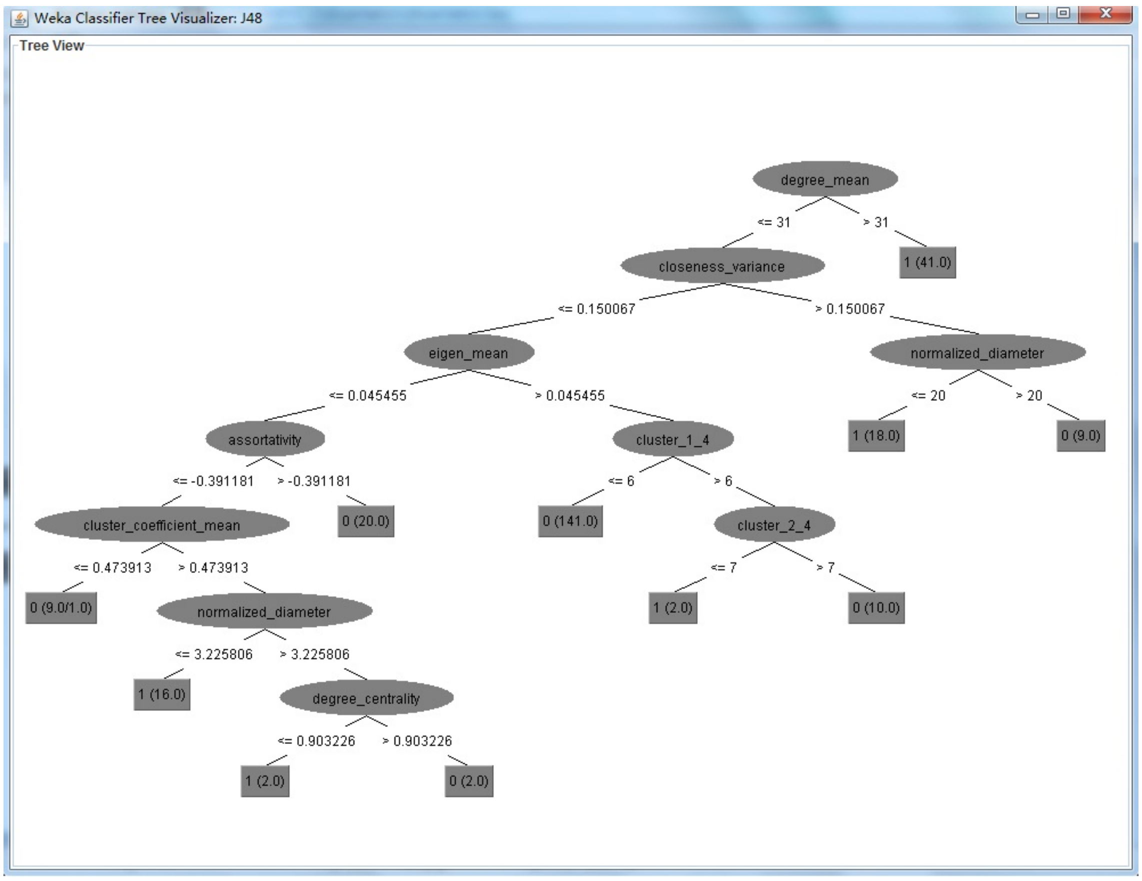


Figure C.1: Illustration of generated decision tree model