面向软件缺陷预测的网络嵌入特征研究

刘靖雯[1], 晋武侠[1,2], 屈 宇[3], 金洋旭[4], 范 铭[1]

[1]西安交通大学 陕西省智能网络与网络安全教育部重点实验室, 西安 中国 710049
[2]西安交通大学 软件学院 西安 中国 710049
[3]美国加州大学河滨分校计算机科学与工程系 河滨 美国加州 92521
[4]中国银行软件中心 西安 中国 710038

摘要 已有研究根据软件的代码依赖、修改历史、协同开发关系等, 建立网络模型来预测软件的缺陷; 近年来, 网络嵌入技术广泛用于软件网络分析, 显著提升了缺陷预测效果。本研究发现不同软件关联网络和网络嵌入算法的组合将影响缺陷预测性能。具体地, 本文针对 3 种软件关联网络(类依赖网络、文件耦合网络和开发者贡献网络), 并应用 6 类网络嵌入方法, 分析不同网络嵌入方法所保持的软件结构特征及其对缺陷预测性能的影响。在 12 个开源 Java 系统上的实验结果显示: 在类依赖网络和文件耦合网络, 传统的度量特征上结合网络嵌入特征后, 缺陷预测效果得到显著提升; DeepWalk、Grarep 和 Node2vec 网络嵌入算法更擅长学习网络的同质性, 缺陷预测效果更好; 网络嵌入特征以及缺陷预测性能对嵌入算法的参数配置比较敏感。本研究结论有助于指导缺陷预测中软件关联网络和网络嵌入方法的选择。

关键词 缺陷预测; 网络嵌入; 软件关联网络
中图法分类号 TP311.53 **DOI 号** 10.19363/J.cnki.cn10-1380/tn.2021.05.03

Research on Network Embedding Features for Software Defect Prediction

LIU Jingwen[1], JIN Wuxia[1,2], QU Yu[3], JIN Yangxu[4], FAN Ming[1]

[1]Key Laboratory of intelligent network and network security, Ministry of Education, Xi'an Jiao Tong University, Xi'an 710049, China
[2]School of Software Engineering, Xi'an Jiao Tong University, Xi'an 710049, China
[3]Department of Computer Science and Engineering, University of California, Riverside CA 92521, USA
[4]Bank of China, Software Center, Xi'an 710038, China

Abstract Researchers predict software defects based on software networks modeled from code dependencies, revision history, and collaborative development. Recently, network embedding techniques have been widely used for software network analysis, significantly improving the defect prediction performance. Our study reveals that different combinations of software associated networks and network embeddings will produce diverse prediction performance. Concretely, this work constructs three kinds of software associated networks (i.e., Class Dependency Network, Change Coupling Network, and Developer Contribution Network), analyzes the software structure preserved by 6 kinds of network embedding methods, and compares their performance in defect prediction. The results on 12 open-source Java systems indicate that, on Class Dependency Network and Change Coupling Network, after the traditional features combined with network embedding features, the defect prediction performance is enhanced significantly; DeepWalk, Grarep and Node2vec are better at learning network homophily, thus producing better prediction performance; the structure features learned by network embedding and their prediction performance by them are sensitive to embedding parameter settings. These results provide guidance in the selection of software associated networks and network embedding techniques for defect prediction.

Key words defect prediction; network embedding; software associated network

通讯作者: 晋武侠, 博士, 助理教授, Email: jinwuxia@mail.xjtu.edu.cn。
本课题得到国家重点研发计划资助项目(No.2018YFB1004500), 国家自然科学基金(No.61632015, No.61772408, No.U1766215, No.61721002, No.61833015, No.62002280, No.61902306, No.61602369), 国网陕西省电力公司科技项目(No.5226SX1800FC), 教育部创新团队(No.IRT_17R86)和中国工程科技知识中心项目, 中国博士后资助项目(No.2019TQ0251, No.2020M673439)的资助。
收稿日期: 2020-06-29; 修改日期: 2020-09-28; 定稿日期: 2021-03-05

1 引言

现代软件系统需求不断增多, 系统日渐复杂, 维护代码变得越来越费时费力, 导致复杂软件系统的缺陷和漏洞难以避免。国内外历史上出现过多次由于软件缺陷而导致的重大事故[1], 预测软件代码缺陷以有效保证软件可信性成为普遍关注的问题。软件缺陷预测[1-4]是使用代码复杂性和变更历史等软件特征构建分类器以预测可能包含缺陷的代码区域的过程, 该技术可提前发现潜在的缺陷和漏洞。近年来, Promise 数据集和美国国家航空航天局项目(NASA 数据集[5])的开源共享进一步推动了学术界在缺陷预测领域上的广泛研究。

根据缺陷预测使用的软件特征的不同, 已有的软件缺陷预测工作可分为基于代码度量特征的预测[6-8]和基于网络度量特征的预测[9-10]。传统代码度量特征主要包括 Halstead 特征[6], 基于依赖性的 McCabe 特征[7], 面向对象程序的 CK 特征[8]等, 但难以刻画软件的结构特征。网络度量特征基于软件实体间的依赖关系[11]构成的网络来度量结构特性, 例如自我中心网络的连接点、边数[4]等等, 但在高维的网络结构空间上度量结构特性, 复杂性较高。

为了降低网络结构特征表示的复杂性, 近年来网络嵌入(Network Embedding)算法研究取得重大进展。网络嵌入[12]的核心思想是, 在保持网络结构特性(一阶相似度、二阶相似度等)的同时, 将高维稀疏的网络结构或者图结构映射到低维稠密的向量空间。这种嵌入后的网络特征便于输入到机器学习模型中。嵌入方法主要包括基于矩阵分解的方法[13-14]、基于随机游走的方法[15-16]、基于深度学习的方法[17]。网络嵌入目前已应用到网络压缩[18-20]、可视化[21-23]、聚类[24]、链路预测[25-27]和节点分类[28]等不同领域。近年来软件关联网络嵌入也开始应用于软件缺陷预测领域, 然而, 不同软件关联网络以及不同网络嵌入算法对缺陷预测性能的影响研究仍然很稀缺。

本文将应用 6 种网络嵌入算法在 12 个开源的 Java 软件系统上, 研究基于网络嵌入特征的缺陷预测, 并在缺陷预测场景中深入分析网络嵌入算法保持的软件结构特征。首先, 从源代码层次、软件演化层次、开发者贡献层次分别构建软件关联网络, 从多层次刻画软件系统的结构。然后, 提出了一种基于网络嵌入和传统度量特征相结合的缺陷预测模型, 并分析不同软件关联网络上的缺陷预测性能。最后, 在缺陷预测场景中深入研究网络嵌入特征, 分析不同网络嵌入算法所保持的软件关联网络结构的差异以及导致缺陷预测性能差异的原因。具体工作如下:

(1) 构建 3 种软件关联网络模型。通过解析软件系统的源代码, 构建类层次依赖网络; 基于修订历史记录中文件的共同修改信息, 构建文件耦合网络; 通过挖掘开发者对程序模块的提交操作, 构建开发者贡献网络。

(2) 设计基于网络嵌入特征的缺陷预测模型。该模型通过网络嵌入算法学习软件关联网络的结构特征, 将高维稀疏网络结构编码至低维稠密向量空间; 并解决缺陷预测中常见的非平衡数据集问题。实验发现, 在传统的代码度量特征和网络度量特征上, 结合网络嵌入特征后, 缺陷预测性能平均提高了至少 2%。

(3) 对比分析 6 种网络嵌入算法在构建的 3 种软件关联网络上通过随机森林、朴素贝叶斯、支持向量机和多层感知器的缺陷预测效果。实验发现: 使用 DeepWalk、Grarep 和 Node2vec 嵌入算法的缺陷预测效果最好, LINE 和 GF 最差; 此外, 同一种算法在类依赖网络和文件耦合网络上的缺陷预测效果比开发者贡献网络要好; 随机森林分类模型可以在任意网络中得到最优的缺陷预测性能。

(4) 比较 6 种网络嵌入算法保持的软件关联网络的结构特性以分析缺陷预测效果存在差异的原因, 并探索网络嵌入算法的不同参数设置对网络结构特征表示以及缺陷预测性能的影响。对网络嵌入算法得到的节点特征向量进行聚类, 度量聚类后的网络结构特征与软件关联网络社区结构的相似性, 并分析相似性与缺陷预测性能之间的相关性。实验发现: 当网络嵌入算法擅长学习网络的同质性时, 可以达到良好的缺陷预测效果; 不同网络嵌入算法得到的聚类结构相似时, 其对应的缺陷预测结果也相近, 例如 DeepWalk 和 Grarep 算法的聚类结果相似, 其缺陷预测结果也相似; 网络嵌入特征和基于网络嵌入的缺陷预测性能对网络嵌入算法的参数配置比较敏感。

其余章节安排。第 2 节介绍相关研究现状; 第 3 节介绍软件关联网络建模和网络嵌入算法, 并给出案例展示; 第 4 节展示基于网络嵌入特征的缺陷预测方法; 第 5 节叙述实验和分析结果, 包括对数据的预处理、缺陷预测结果分析、网络嵌入特征分析、以及参数影响分析; 第 6 节总结本文工作并展望未来研究方向。

2 相关工作

2.1 软件缺陷预测

软件缺陷预测是软件工程领域研究的热点。软

件缺陷预测通过分析和挖掘软件库(例如项目托管的缺陷跟踪系统和版本控制系统), 根据软件源代码和开发过程设计与软件缺陷相关的度量指标, 构建缺陷预测模型, 使用训练出的模型识别出被测项目内可能存在缺陷的模块。开发人员可对这些潜在的缺陷模块投入更多的精力进行软件测试, 修改或完善代码以保证软件产品质量。Promise 数据集的建立和美国国家航空航天局项目(NASA 数据集[5])的开源共享推动了缺陷预测的研究进展。影响软件缺陷预测性能的主要因素包括: 度量指标的设计、预测模型的构建和数据集相关问题[30]。在预测模型中, 机器学习模型是目前主流的方法[31-32]。除此之外, 也有基于缓存的方法[33-34], 利用缺陷局部性原理识别缺陷。对于数据集, 经常会出现噪音[35]和非平衡[36-37]等问题, 目前也已经有针对这些问题的研究。

设计出与缺陷具有强相关性的度量指标是提高预测效果的关键。软件缺陷预测用到的特征常通过度量获取。传统的软件度量指标包括基于运算符和操作数的 Halstead 特征[6], 基于依赖性的 McCabe 特征[7], 面向对象程序的 CK 特征[8]等。同时, 也有研究表明, 软件演化数据在缺陷预测方面比代码度量更有效[38], 在缺陷预测中利用软件演化数据的一个方面是将变更过程建模为网络[39], 并使用不同的网络度量(例如, 度统计或中心度量)改进机器学习分类器的性能。其中 Zimmermann 和 Nagappan[4]提出在缺陷预测中使用软件系统依赖关系图上的网络度量。Ma 等人[10]进一步评估了网络度量在缺陷预测中的预测有效性。Chen 等人[9]利用网络度量来预测有严重缺陷的软件, 他们发现大多数网络与严重的缺陷显著相关。另一方面是从代码修改[40-42], 开发人员经验[43-45], 程序模块间的依赖性[4,46]或项目团队组织构架角度[47-49]等方面出发来设计度量指标。程序的良好的语法定义、隐藏在抽象语法树中的丰富语义以及程序中的依赖网络信息通常无法被这些指标很好的捕获。因此传统方法的预测结果差强人意。

2.2 软件关联网络

软件关联网络主要是根据实体和实体之间的依赖关系构建的, 从各种不同的粒度(包、类、方法、函数)可对软件网络进行拓扑分析。软件关联网络产生于复杂网络理论, 复杂网络[50-51]最初是在物理学和数据科学中开发的, 并在不同领域被广泛采用。近年来, 该理论已成功地应用于软件工程领域。Myers CR[52]提出了复杂网络理论在软件系统中的应用。Concas 等人[53]和 Louridas 等人[54]随后在软件系统中观察到了幂律分布。

软件网络被用于软件演化过程的建模和理解[55-56], 软件演化的预测[57], 软件结构的分析和评估[52,54], 软件执行过程和行为建模[58], 软件的质量评估[57,59-60], 软件的安全分析[61-63]。其中 Li[55]提出了一种软件网络演化的模块化连接机制。Subelj 等[64]分析了类依赖网络中存在的社区结构。Qu 等[65]发现软件中方法和它们之间的调用关系呈现出相对显著的社区结构, 并基于软件网络的社区划分提出了两个类内聚的度量指标。Pan 等 [66]从实际的 Java 软件系统中构造了一组加权软件网络, 并利用加权 k 核分解对其拓扑性质进行了实证研究。Cai[58]将软件执行过程建模为演进的复杂网络。Bhattacharya[57]通过源代码库和缺陷追踪系统中的实体关系构建网络以更好地理解软件演化过程, 并构建预测器以促进软件开发和维护。Concas 等[59]研究了 Java 软件网络中划分的社区与缺陷之间的关系。Pan 等[60]提出了一个针对软件网络的度量元来量化软件的稳定性。

2.3 网络嵌入算法及其应用

复杂网络结构中节点和边众多, 拓扑结构复杂, 难以高效分析。近几年出现的网络嵌入算法[13-17]可以有效解决网络数据难以分析的难题, 其中心思想就在保持网络结构特性(一阶相似度、二阶相似度等)的同时将高维稀疏的网络结构或者图结构映射到低维稠密的向量空间。

表 1 常用的网络嵌入算法

Table 1 Common Network Embedding Algorithms

分类	算法名称	保留结构特征	时间复杂度
随机游走	DeepWalk(2014)	1-k 阶相似度	$O(\|V\|)$
	Node2vec(2016)	1-k 阶相似度	$O(\|V\|)$
矩阵分解	GraRep(2015)	1-k 阶相似度	$O(\|V\|^3)$
	Graph Factorization(2013)	1 阶相似度	$O(\|E\|d)$
深度学习	SDNE(2016)	1 阶/2 阶相似度	$O(\|V\|\|E\|)$
综合	LINE(2015)	1 阶/2 阶相似度	$O(\|E\|d)$

LE(Laplacian Eigenmaps)[64]方法和 LLE(Locally Linear Embedding) [65]方法是传统的将网络节点映射为低维向量的方法。然而, 这些方法并没有很好的可扩展性。近些年又提出了很多的网络嵌入算法, 根据其主要性质可以分为 3 大类: 基于矩阵分解[13-14]、基于随机游走[15-16]、基于深度学习[17]。随机游走已经被用于近似网络的许多属性, 包括节点中心性和相似性, 适用于过大的无法完整测量的网络; 基于分解的算法以矩阵的形式表示节点之间的连接, 并将矩阵分解以获得向量表示; 基于深度学习的模型可

以对数据中的非线性结构进行建模, 捕获网络中非线性的特征。表 1 展示了每一类中的代表性算法以及它们保留的结构特征和时间复杂度。

网络嵌入算法是复杂网络分析的研究重点, 也是将深度学习应用到网络分析的有效途径, 网络嵌入算法学习到的特征表示可以用作基于图的各种任务的特征, 例如分类、聚类、链路预测和可视化。在软件工程领域, 由于软件内部存在复杂的调用以及依赖关系, 软件中存在多种类型的网络, 所以用网络嵌入算法可以解决很多软件中存在的问题。GefGroid[69]是基于网络嵌入算法的安卓恶意软件家族分析方法, 根据函数调用图划分出子图, 用 struc2vec[70]将子图中 API 调用节点表示为向量, 通过向量相似度的计算聚类出安卓恶意家族。Node2defect[71]利用 Node2vec[16]将软件中的依赖网络节点转化为向量, 用于软件的缺陷预测。Liu 等[72]人利用 DeepWalk[15]获取软件系统结构特征, 并结合软件抽象语法树的语义特征共同进行软件的缺陷预测。Ling 等人[73]提出基于图嵌入的软件项目源代码检索方法。该方法利用 LINE[29]将代码结构图中的节点转化为向量, 用户输入自然语言问题即可检索并返回相关的 API 及其关联信息构成的连通代码子图。此外, 一些工作没有利用常见的网络嵌入算法, 而是通过构造目标函数将网络节点转化为低维的向量, 从而进行方法名或类名的推荐[74-75]。

3 软件关联网络与网络嵌入

本章将构建类依赖网络、文件耦合网络、开发者贡献网络等软件关联网络, 介绍 6 种常用的网络嵌入算法, 最后在一个真实软件系统上展示软件关联网络及其嵌入特征。

3.1 软件关联网络建模

本节将从源代码、演化历史中的共同修改、演化历史中的开发者贡献 3 个层次对软件结构进行建模, 生成软件关联网络。

3.1.1 类依赖网络 (Class Dependency Network, CDN)

从软件的源代码抽取类依赖网络[76]: 对于面向对象系统 P, 其类依赖网络 $CDN_P=(V,E)$ 是有向网络, 其中 V 为 CDN_P 顶点的集合, 每个顶点 $v\in V$ 代表软件系统中的类/接口, E 是网络的边集合, 代表类/接口之间的依赖关系。若类/接口 v_1、v_2 之间存在以下任意一种依赖关系, 则存在边 $e=(v_1,v_2)\in E$:

1) 继承依赖(inheritance): 类 v_1(称为子类、子接口)继承类 v_2(称为父类、父接口)的功能, 并可扩展增加新功能。继承是类与类或者接口与接口之间最常见的依赖关系, 在 Java 中通过关键字 extends 标识;

2) 接口实现依赖(interface implementation): 类 v_1 实现接口 v_2 的功能, 在 Java 中此类关系通过关键字 implements 标识;

3) 方法调用依赖(aggregation): 类 v_1 中的方法调用了类 v_2 中的方法;

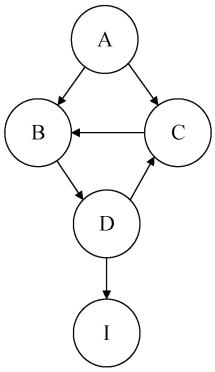
4) 返回值依赖(return types): 类 v_1 中的方法返回了类 v_2 的对象;

5) 参数依赖(parameter types): 类 v_1 的对象作为类 v_2 中的方法的参数。

以图 1 为例, 图 1(a)展示了 Java 类和接口的代码片段, 从中抽取出的类依赖网络见图 1(b)。其中类 A 的方法调用了类 B 的方法, 因此存在一条从节点 A 指向节点 B 的边。

```
interface I {
    public int add(int i);
}

public class A {
    public static void main(String[] args) {
        B b = new B();
        System.out.println(b.counter);
        b.m1();
        C c = new C();
        c.m2(b);
    }
}

class B {
    public int counter=1;
    private D d = new D();
    public void m1() {
        counter=d.add(counter);
        System.out.println(counter);
    }
}

class C {
    public void m2(B b) {
        b.counter=b.counter+1;
        System.out.println(b.counter);
    }
}

class D extends C implements I {
    public int add(int i) {
        return i+1;
    }
}
```

(a)



(b)

图 1 类依赖网络示例

Figure 1 Class Dependency Network

3.1.2 文件耦合网络 (Change Coupling Network, CCN)

本文提出从软件演化历史(记录于软件的版本控制系统如 Git)中的共同修改记录来构建文件耦合网络: 对于面向对象系统 P, 其文件耦合网络 $CCN_P=(V,E)$ 是无向带权图, 其中 V 为图 CCN_P 的顶点集合, 每个顶点 $v\in V$ 代表软件系统中的源文件, E 是图的边集合, 代表源文件在修订历史记录中的的共同修改(co-change)关系, 权值代表源文件间的共同修改频次。

图 2(a)示例出软件修订历史中的 4 次提交记录,

例如, A.java 和 C.java 在 Commit 1 和 Commit 3 中都同时修改, 所以在 CCN 中 A.java 和 C.java 之间存在一条边且权值为 2。图 2(a)对应生成的文件耦合网络见图 2(b)。

构建文件耦合网络需设置**过滤阈值**参数, 该参数是指一次提交所涉及的最大文件数。当一次提交记录中涉及的文件很多时, 相应的文件耦合网络会存在很多权值较小的边, 这种“大提交”往往是由一些通用修改导致, 比如 license 修改, 分支合并等。为了避免过耦合, 需要设置适当的过滤阈值参数以筛选出有效的用于构建文件耦合网络的提交记录。

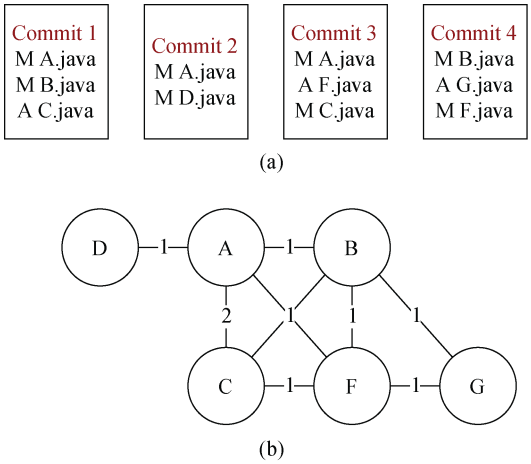


图 2　文件耦合网络示例

Figure 2　Change Coupling Network

3.1.3　开发者贡献网络(Developer Contribution Network, DCN)

从软件演化历史中的开发者维护活动来构建开发者贡献网络 DCN [43]: 对于面向对象系统 P, 其开发者贡献网络 $DCN_P=(D,B,E)$ 是个无向带权二分图。其中 D,B 为图 DCN_P 的顶点的两个集合, 每个顶点 $d\in D$ 代表软件开发者, $b\in B$ 代表类/接口; $E\subseteq\{(d,b)\mid d\in D\wedge b\in B\}$ 是图的边集合, 边(d,b)代表开发者 d 对类/接口 b 进行了提交操作; 边的权重表示开发人员对类/接口的提交次数。

图 3 是开发者贡献网络的示例。其中, 矩形代表类/接口, 人形图案代表开发者, 边代表开发人员对类/接口的修改提交操作, 边的标签代表开发人员提交该文件的次数。例如, 边(d_5,b_1)表示开发人员 d_5 对类 b_1 做了 6 次提交操作。文献[43]研究发现, 开发者贡献网络中中心性越高的类/接口越易发生缺陷(error-prone)。当一个类/接口被较多开发者修改, 则中心性会增强, 所以类 b_1 和类 b_2 比类 b_3 发生缺陷的概率更大; 加之, 开发人员同时处理许多其他文件则中心性会增加, 所以类 b_1 比类 b_2 更有可能发生缺陷。

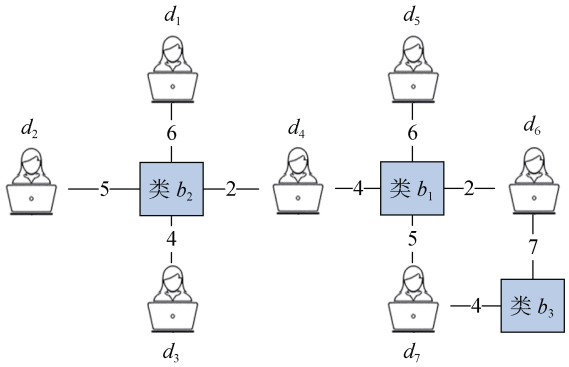


图 3　开发者贡献网络示例

Figure 3　Developer Contribution Network

3.2　网络嵌入算法

网络嵌入[12]的核心思想是在保持网络结构特性(一阶相似度、二阶相似度等)的同时将高维稀疏的网络结构或者图结构映射到低维稠密的向量空间。本文选取 DeepWalk、Node2vec、Grarep、Graph Factorization、SDNE 和 LINE 算法共 6 种代表性的网络嵌入算法, 对软件关联网络进行特征表示。

1) DeepWalk. 由 Perozzi 等人[15]于 2014 年提出, 是最早的基于 Word2vec 的节点向量化模型, 网络中节点的邻域的概念可以使用自然语言中单词上的滑动窗口来定义。主要思路是利用网络中随机游走得到的节点序列, 生成节点的向量表示。DeepWalk 发现网络中随机游走序列的节点分布类似于自然语言中的单词分布。因此对于网络中的每个节点, 生成长度为 t 的随机游走序列。然后用 Skip-gram 模型训练模型, 最大化随机游走序列中节点的概率:

$$\max_{\phi}\Pr(\{v_{i-w},\cdots,v_{i+w}\}\setminus v_i\mid\phi(v_i))$$

其中 w 是窗口大小, $\phi(v_i)$ 是节点 v_i 的表示向量, $\{v_{i-w},\cdots,v_{i+w}\}\setminus v_i$ 是节点 v_i 的上下游节点。

DeepWalk 算法充分利用了网络结构中的随机游走序列的信息, 使用随机游走序列的信息只依赖于局部信息, 所以适用于分布式和在线系统。

2) Node2vec. 该算法通过改变随机游走序列生成的方式进一步扩展了 DeepWalk 算法, 由 Grover 于 2016 年提出[16]。DeepWalk 随机地选取随机游走序列中下一个节点, 而 Node2vec 将广度优先搜索(BFS)和深度优先搜索(DFS)引入随机游走序列的生成过程, 通过两个参数 p、q 实现二者的平衡, 可以同时考虑到局部和全局的信息, 并且具有很高的适应性。

Node2vec 中用参数 p 和 q 控制着行走过程中的转移概率。参数 p 为返回参数(return parameter), 若 p 值高, 可以保证在下一步采样已访问过的节点的可能性比较低; 参数 q 为输入-输出参数(in-out parameter), $q>1$时随机游走会选择离当前节点更近的节点, 以此达到接近 BFS 的效果; $q<1$时随机游走会选择离当前节点更远的节点, 达到类似 DFS 的效果。它的目标函数如下所示:

$$\max_f \sum_{u\in V} \log P(N_s(u)\,|\,f(u))$$

本式含义与 DeepWalk 类似, 其中 $f(u)$ 为当前节点 u 在嵌入空间的向量表示, $N_s(u)$ 为当前节点的上下游节点。

3) Grarep. 是基于矩阵分解的网络嵌入算法之一, 由 Cao 等人[13]在 2015 年提出。Grarep 通过操纵不同的全局转移矩阵, 从图形中直接捕获不同的 k 阶关系信息, 不需要缓慢且复杂的采样过程。与其他算法不同的是, 该算法定义了不同的损失函数, 来捕获不同的 k 阶关系信息, 允许集成不同局部关系信息的非线性组合。

该算法将节点的转移概率定义为 $A=\boldsymbol{D}^{-1}\boldsymbol{S}$, 其中 $\boldsymbol{D}$ 为度矩阵, $\boldsymbol{S}$ 为邻接矩阵。利用 A 可以计算出 k 阶转移概率 A^k。将将其中从起始点 w 到终止点 c 的 k-step 路径记作(w, c)。优化目标是: 最大化(w, c)所代表的路径在图中的概率; 最小化除了(w, c)以外的路径在图的概率来保留 k 阶相似度。Grarep 的主要缺点是其可扩展性。并且, 该算法在学习过程中将奇异值分解方法(SVD)应用于矩阵的幂以获得节点的低维表示, 这可能会导致高昂的计算成本。

4) Graph Factorization. 由 Ahmed 等人[14]在 2013 年提出, 后面简称 GF。目前来说, GF 是第一个将算法复杂度降到 $O(|E|)$的网络嵌入算法。为了获得表征结果, GF 对网络的邻接矩阵进行因式分解, 最小化随后的损失函数如下所示:

$$\phi(Y,\lambda)=\frac{1}{2}\sum_{(i,j)\in E}(W_{ij}-\left\langle \boldsymbol{Y}_i,\boldsymbol{Y}_j\right\rangle)^2+\frac{\lambda}{2}\sum_i \|\,Y_i\,\|^2$$

式中 W_{ij} 是节点 i 和节点 j 连接边的权重, $<\boldsymbol{Y}_i,\boldsymbol{Y}_j>$ 是降维后节点向量的内积, λ 是正则化系数。值得注意的是, 这里的求和是针对观察到的边而非所有边, 这是为了算法的可扩展性, 但这样可能会在结果中带来噪声。即使表示向量维度为|V|, 该损失函数的最小值仍然有可能大于 0。

5) SDNE. 由 Wang 等人[17]在 2016 年提出, 使用深度自动编码器来同时保留一阶和二阶相似度的算法。一阶相似度衡量的是相邻的两个顶点对之间相似性。二阶相似度衡量的是两个顶点的邻居集合的相似程度。该算法通过保持网络的一阶相似度来保持网络局部结构, 通过保持网络的二阶相似度来保持全局的网络结构。论文采用了半监督结构, 其中的无监督组件(编码器、解码器)通过重构邻接矩阵来保持网络的全局结构, 具有相似邻域结构的顶点将被映射到相近的向量空间; 而监督组件借鉴了 Laplacian Eigenmaps 算法, 利用一阶相似度来保持局部的网络结构, 使得彼此相连的节点在映射空间中也十分靠近。由于模型高度非线性, 在参数空间中会存在很多局部最优解。为了解决该问题, 该算法采用 Deep Belief Network 来对参数进行预先训练, 完整算法可参考论文原文。SDNE 的主要贡献在于提出了一种新的半监督深层学习模型, 结合一阶与二阶相似度的优点, 用于表示网络的局部结构属性和全局结构属性。

6) LINE. 由 Tang 等人[29]在 2015 年提出, 该方法的可扩展性很好,能够快速地对大规模网络进行嵌入,并且能够同时保留局部和全局的结构信息。它通过定义两个函数来计算节点的嵌入向量, 分别用于一阶相似度和二阶相似度的计算, 并最小化这两个函数的组合。一阶相似度的函数类似于 GF 算法, 因为它们都旨在保持邻接矩阵和降维后节点的点积接近。不同的是, GF 通过直接最小化两者的差异来做到这一点, 而 LINE 用邻接矩阵和嵌入矩阵分别定义了每对节点的两个联合概率分布, 然后最小化这两个分布的 Kullback-Leibler(KL)散度。这两个分布如下所示:

$$p_1(v_i,v_j)=\frac{1}{1+\exp(-\left\langle \boldsymbol{Y}_i,\boldsymbol{Y}_j\right\rangle)}$$

$$P_1(v_i,v_j)=\frac{W_{ij}}{\sum_{(i,j)\in E}W_{ij}}$$

该算法的目标函数如下所示:

$$O_1=KL(P_1,p_1)$$

其中 v_i,v_j 为网络中的两个节点, $\boldsymbol{Y}_i,\boldsymbol{Y}_j$ 是 v_i,v_j 的嵌入向量, W_{ij} 是 v_i,v_j 的连接权重。作者类似地定义了对于二阶相似度的概率分布和目标函数。该算法适用于任意类型的信息网络, 并能够轻易拓展到百万个节点。

3.3 案例展示

本节展示Lucene软件[①]的3种软件关联网络及其网络嵌入结果。

图 4 可视化出 3 种软件关联网络: 使用工具Understand[②]抽取类依赖关系, 构建类依赖网络 CDN, 见图 4(a); 基于 Git log 中文件的共同修改记录, 构建的文件耦合网络 CCN, 见图 4(b); 通过挖掘开发者对文件的修改提交操作, 构建开发者贡献网络 DCN, 见图 4(c)。在 3 种软件关联网络中, 红色节点表示存在缺陷的类/接口/文件, 蓝色节点表示不存在缺陷的类/接口/文件, 绿色节点表示开发人员。从图中可以看出, 在该软件中存在缺陷的类/接口/文件相对较多, 且缺陷节点的分布没有明显规律。

选取 Grarep 和 LINE 网络嵌入算法将 Lucene 软件的 CDN 的节点转化为向量表示。为了观察向量反应出的网络节点之间的关系, 与文献[16]类似, 我们使用 k 均值聚类算法[84](k-means clustering algorithm)基于嵌入后的向量信息对网络节点进行聚类, 使距离相近的向量被划分到同一个簇(cluster)中, 不同簇使用不同颜色渲染, 结果如图 5 所示, 图 5(a)为 Grarep 特征的聚类结果, 图 5(b)为 LINE 特征的聚类结果。可以观察到, Grarep 算法和 LINE 算法嵌入后的节点向量经聚类得到的结果存在很大差异, 初步得出不同网络嵌入保持的网络结构特征可能不同。

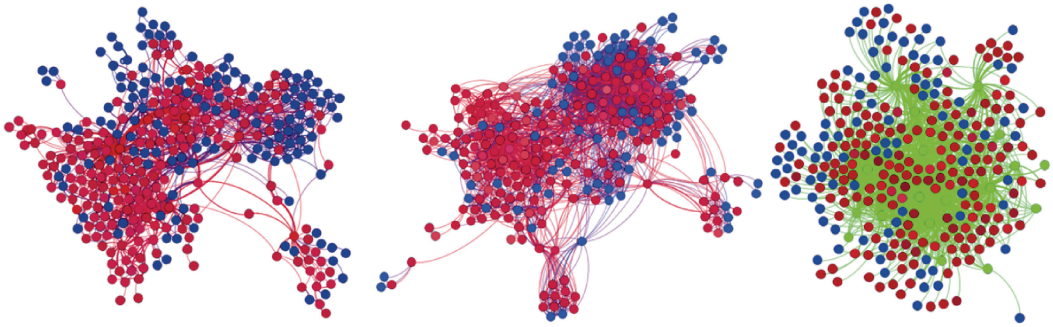


图 4 Lucene 的软件关联网络

Figure 4 Software Associated Network of Lucene

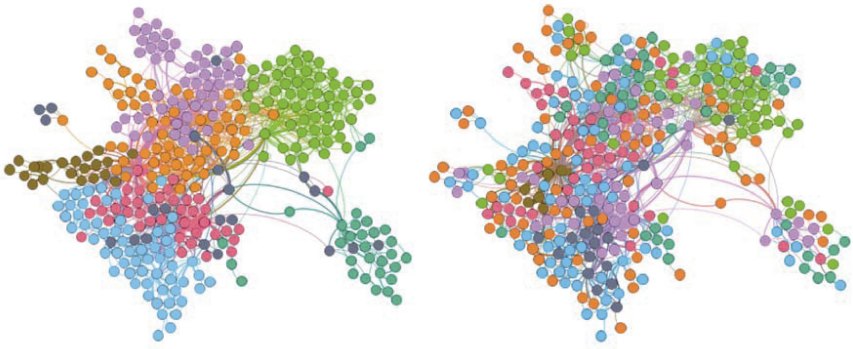


图 5 Lucene 软件 CDN 的网络嵌入特征的聚类结果

Figure 5 Clusters of Class Dependency Network Embedding Features for Lucene

4 基于软件关联网络嵌入的缺陷预测

本文设计了一种基于软件关联网络嵌入特征的缺陷预测方法, 该方法除了使用代码度量和网络度量的软件特征, 也使用了软件关联网络嵌入特征。本节先介绍方法流程, 再详细介绍每个步骤。

4.1 方法流程

图 6 展示了基于软件关联网络嵌入的缺陷预测

① https://lucene.apache.org/

② https://scitools.com/

流程:

1) 输入处理。根据软件的源代码和修订历史构建出 CDN、CCN、DCN 共 3 种软件关联网络。

2) 特征提取。从源代码和软件关联网络中分别提取出 3 种特征: 代码度量特征(CM)、网络度量特征(NM)、网络嵌入特征(NE)。其中 CM 和 NM 属于传统的度量特征。

3) 模型训练。对缺陷数据集使用 SMOTUNED 算法[77]进行非平衡数据的处理, 将上一步得到的 3 种特征和对应的缺陷标签输入随机森林、支持向量机、朴素贝叶斯和多层感知器进行模型训练并测试, 预测是否存在缺陷。

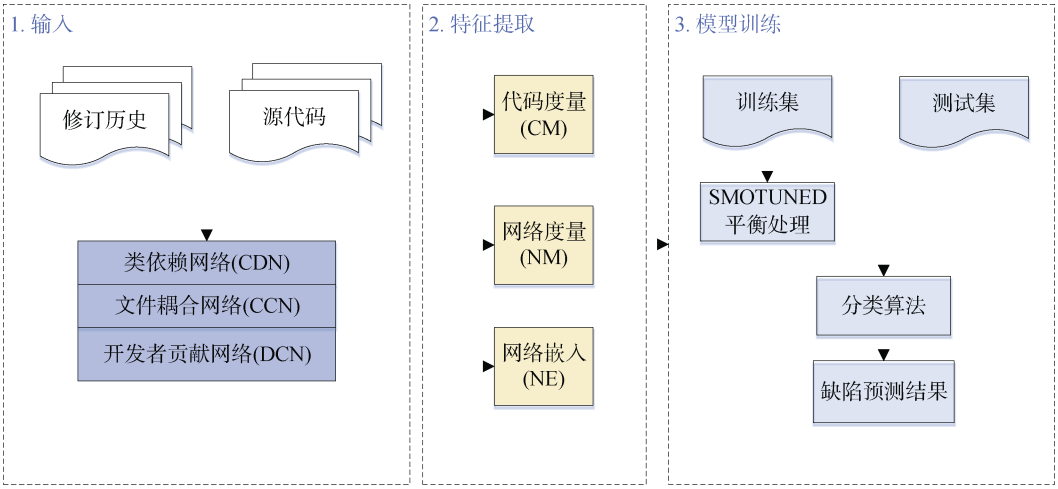


图 6 基于软件关联网络嵌入的缺陷预测方法流程

Figure 6 Defect Prediction Based on Network Embedding

4.2 特征提取

本节详细介绍从源代码和软件关联网络得到的度量特征:

(1) 代码度量特征

本文将基于源代码的度量结果作为代码度量特征。已有研究提出了各种代码度量指标, 本文使用了常用的 C&K(Chidamber-Kemerer)指标[8], McCabe 的度量[7](Max_CC, Avg_CC)等共计 20 个代码度量指标。这些指标的集合记为 CM(Code Metrics), 来源于公开可用的 tera-PROMISE 数据库, 表 2 展示指标的名称和说明。

(2) 网络度量特征

本文使用了 59 种基于网络的指标, 这些指标集合记作网络度量 NM(Network Metrics), 包括对自我中心网络(Ego Network)的度量和对全局网络(Global Network)的度量。这些指标详见 Zimmermann 等[4]和 Ma 等[10]的研究工作, 其中 16 个指标的说明见表 3。NM 指标可以使用 Ucinet①来计算, Ucinet 可以处理矩阵格式的网络结构数据, 探测网络结构的度量指标。

表 2 代码度量(CM)和说明

Table 2 Code Metrics and Descriptions

指标名称	说明	指标名称	说明
AMC	Average Method Complexity	LCOM	Lack of Cohesion in Methods
AVG_CC	Average McCabe	LCOM3	Lack of Cohesion in Methods 3
Ca	Afferent Couplings	LOC	LINEs of Code
CAM	Cohesion Among Methods of Class	Max_CC	Maximum McCabe
CBM	Coupling Between Methods	MFA	Measure of Function Abstraction
CBO	Coupling Between Object Classes	MOA	Measure of Aggregation
Ce	Efferent Couplings	NOC	Number of Children
DAM	Data Access Metric	NPM	Number of Public Methods
DIT	Depth of Inheritance Tree	RFC	Response for a Class
IC	Inheritance Coupling	WMC	Weighed Methods per Class

① www.analytictech.com/ucinet/

表 3 网络度量(NM)和说明

Table 3 Network Metrics and Descriptions

指标名称	说明	指标名称	说明
Size	Size of ego network	Broker	Number of pairs not directly connected
Ties	Number of directed ties	nBroker	Broker normalized by the number of pairs
Pairs	Number of ordered pairs	ReachEffic	Two Step Reach divided Size
Density	Ties divided by pairs	EgoBetweenness	Betweenness of ego in own network
Degree	Degree centralization	nWeakComp	Number of weak components
AvgDist	Average geodesic distance	pWeakComp	NweakComp divided by Size
Diameter	Longest distance in egonet	InCloseness Centrality	Countdown of the in-distances
Two_Step_Reach	Number of nodes within 2 links	OutCloseness Centrality	Countdown of the out-distances

(3) 网络嵌入特征

网络嵌入特征反映软件关联网络的结构信息, 本文分别使用 DeepWalk、Node2vec、Grarep、Graph Factorization、SDNE 和 LINE 算法, 将软件关联网络中的节点表示为反映网络结构信息的向量。

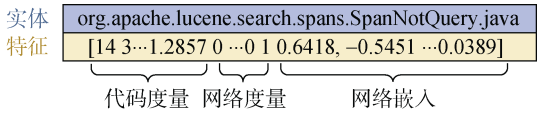


图 7 3 种特征合并后的特征向量示例

Figure 7 The Feature Vector with 3 Features

将该向量与此节点的代码度量和网络度量值合并后的向量 , 和对应的缺陷标签(节点对应的类/文件/接口是否存在缺陷)作为模型训练过程的输入。如图 7 所示, 对于 Lucene 软件中的一个节点 org.apache.lucene.search.spans.SpanNotQuery.java, 使用 DeepWalk 算法, 三种特征合并后的向量为: 对该节点的代码度量、网络度量与基于 DeepWalk 得到的节点向量的合并。

4.3 模型训练

在模型训练过程中, 将特征向量和对应缺陷标签输入至机器学习模型中, 利用十次三折交叉验证训练缺陷预测模型。大多数训练集中缺陷和非缺陷的数据比例不平衡, 因此需要先平衡训练集, 然后输入至随机森林、多层感知器、支持向量机、朴素贝叶斯等 4 种分类器中进行训练。

4.3.1 非平衡数据集处理

很多机器学习算法都假设训练数据集是均匀分布的, 若把这些算法直接应用, 大多数情况下无法取得理想的结果, 这是因为实际数据往往分布的很不均匀, 存在长尾现象: 即数据中的一类样本在数量上远多于另一类。这种情况下, 传统机器学习训练算法的结果可能会偏向于多数类, 对于少数类的预测性能会很差。为了解决数据非平衡问题, He 等人[78], Zhang 等人[79]和 Lopez 等人[80]提出了处理非平衡数据的模型。包括: 模型融合(Ensemble)方法, 代价敏感分类(Cost-sensitive)方法, 基于核的算法(Kernel-based)和主动学习(Active Learning)算法等。

本文使用 Agrawal 等人[77]在 2018 年提出的 SMOTUNED(Autotuning Version of SMOTE)算法来对实验数据进行平衡处理。SMOTUNED 算法是对 Chawala 等人[81]在 2002 年提出的 SMOTE(Synthetic Minority Over-sampling Technique)算法的改进。SMOTE 通过对少数类样本进行分析和模拟, 将新样本添加到数据集中来减少数据失衡程度。该方法从少数类中的每个样本 x 的 k 个近邻(欧氏距离)中随机挑选 N 个样本进行随机线性插值, 构造新的少数类样本, 并将新样本和原数据合成得到新的训练集。它是基于随机过采样的一种改进方案, 是目前处理非平衡数据的常用手段。但是, SMOTE 算法中涉及很多参数, 参数的设置往往依赖于人们的经验, 有些研究使用了该算法的默认参数[77], 这样的设置并不能很好的适应所有数据集。SMOTUNED 使用差分进化算法(Differential Evolution), 随机生成 SMOTE 算法中参数的初值, 通过变异、交叉、选择的演化过程逐步自适应地寻求参数的最优解。

4.3.2 随机森林

随机森林(Random Forest)[82]是利用多棵树对样本进行训练并预测的一种分类器, 可以基于开源机器学习框架 scikit-learn①实现, 并利用 scikit-learn 的网格搜索(grid-search)功能调整随机森林的参数。随机森林以随机的方式建立包含多棵决策树的森林, 根据决策树的分类结果对每个记录进行投票表决决定最终的分类。随机森林可用于处理分类和回归问

① https://scikit-learn.org/stable/

题。随机森林的随机性可以有效地防止过拟合情况的出现, 同时多棵决策树增强了模型的泛化能力。

4.3.3 多层感知器

多层感知器(Multi-Layer Perception, MLP)是一种基于误差反馈的人工神经网络, 由一个输入层、一个或多个隐层、一个输出层构成。MLP 神经网络不同层之间是全连接的。训练过程包括正向和反向两种传播过程。在正向传播过程中得到输出结果, 在反向传播过程中进行验证, 根据误差逐层调整各层神经元的权值, 最终使输出结果满足要求。

4.3.4 支持向量机

支持向量机(Support Vector Machine, SVM)是一个利用有监督学习方法对数据进行二元分类的广义线性分类器。SVM 的基本思想是在分类问题中求解能够正确划分训练数据集并且几何间隔最大的超平面。一些非线性可分的问题可以通过核函数替换实例和实例中的内积解决, 常用的核函数有: 高斯核函数、Sigmoid 核函数和多项式核函数。本文的输入特征线性不可分, 因此选择高斯核函数。

4.3.5 朴素贝叶斯

朴素贝叶斯(Naive Bayesian, NB)是基于贝叶斯定理与特征条件独立假设的分类方法。通过贝叶斯公式可以由已知的先验概率和条件概率, 推算出后验概率 $P(B|A)$。而朴素贝叶斯算法对条件概率分布做出了独立性的假设, 在这个假设的前提上, 结合贝叶斯定理即可得到分类结果。朴素贝叶斯有三个常用模型: 高斯、多项式、伯努利, 可以根据实际情况选择不同的模型。多项式朴素贝叶斯模型常用于文本分类问题中, A 中的特征为单词在文本中出现的次数。如果 A 中的特征是连续变量, 则假设在 B 的条件下, A 服从高斯分布, 则使用高斯朴素贝叶斯模型。由于本文中的输入特征均为连续变量, 因此选择高斯朴素贝叶斯模型。

5 实验

本章首先介绍选取的数据集及其预处理, 然后分析基于软件关联网络嵌入的缺陷预测结果, 紧接着分析网络嵌入保持的网络结构特征以及缺陷预测性能差异的原因, 最后分析不同参数配置下的网络嵌入特征对缺陷预测性能的影响。

5.1 数据集预处理

5.1.1 实验对象

本文使用了 12 个常用的大型开源 Java 软件系统作为实验对象。这些系统的缺陷数据来自于公开可用的 tera-PROMISE 数据库①和 D'Ambros 等人[83]于 2010 年发布的缺陷数据集②。这 12 个软件系统包括: Ant 是用于自动化软件构建过程的命令行工具; Camel 是集成框架; Ivy 是传递依赖管理器; Lucene 是搜索框架和算法; Poi 是用来处理 Microsoft Office 文件的库; Synapse 实现了高性能的企业服务总线(ESB); Tomcat 是 web 服务器, Servlet 和 JSP 容器; Velocity 是模板引擎, 用于引用 Java 代码中定义的对象; Xalan 用于处理 xml 文档; Equinox Framework 是 OSGi 核心框架规范的实现; JDT Core 是 Eclipse IDE 的核心组件; PDE UI 也是 Eclipse 组件, 提供了一套全面的工具来开发和测试 Eclipse 插件和其他产品。

这些系统及其软件关联网络的统计信息如表 4 所示。表中 *SLOC* 列表示软件的源代码行数; $|C_{RD} \cap C_{CDN}|$ 代表了既出现在 CDN 中又在缺陷数据集中的实体的数量; $|C_{RD} \cap C_{CCN}|$ 代表了既出现在 CCN 中又在缺陷数据集中的实体的数量; $|C_{RD} \cap C_{DCN}|$ 代表了既出现在 DCN 中又在缺陷数据集中的实体的数量; $P_{CDNDefect}$ 表示 CDN 中被标记为缺陷的实体所占的百分比, $P_{CCNDefect}$ 表示 CCN 中被标记为缺陷的实体所占的百分比, $P_{DCNDefect}$ 表示 DCN 中被标记为缺陷的实体所占的百分比。

5.1.2 参数设置

(1) 软件关联网络的过滤阈值

如前文所述, 构建 CCN 和 DCN 时需要设置过滤阈值, 过大或过小的过滤阈值都不能有效反映耦合信息。本实验将以 CCN 为例, 使用从 15 到 35、间隔为 5 的不同过滤阈值进行预处理实验, 也对不设置过滤阈值 (即过滤阈值为 ∞)进行实验, 目的是验证不同过滤阈值对缺陷预测实验效果的影响, 同时找出最适用于缺陷预测模型的过滤阈值用于构建软件关联网络。过滤阈值为 n 意味着构建网络时忽略涉及的文件数超过 n 的历史修改提交记录。表 5 展示了不同过滤阈值下 CCN 的统计信息, N_{CCN} 代表节点数(节点代表的实体不一定在缺陷数据集中), E_{CCN} 代表边数。

① http://openscience.us/repo/

② http://bug.inf.usi.ch/index.php

表 4 实验对象统计信息

Table 4 Dataset Summary

软件名称	版本	*SLOC*	$\lvert C_{RD} \cap C_{CDN} \rvert$	$P_{CDNDefect}$ (%)	$\lvert C_{RD} \cap C_{CCN} \rvert$	$P_{CCNDefect}$ (%)	$\lvert C_{RD} \cap C_{DCN} \rvert$	$P_{DCNDefect}$ (%)
Ant	1.7.0	93520	729	22.8	690	23.6	712	23.0
Ivy	2.0	36636	349	11.5	296	13.5	303	13.2
Camel	1.6.0	98962	902	20.8	871	20.7	880	21.0
Lucene	2.4.0	35984	337	60.2	272	67.2	262	67.6
Poi	3.0	53097	439	64	273	61.5	301	62.5
Synapse	1.2	46060	250	34	237	35.4	238	35.3
Tomcat	6.0.38	166396	744	10.2	466	15.0	490	14.7
Velocity	1.6.1	26636	228	34.2	218	35.3	224	34.8
Xalan	2.6.0	155067	860	45.5	614	50.2	627	50.4
JDT Core	3.4	311316	719	26.7	720	26.7	720	26.7
PDE UI	3.4.1	234608	1101	17.3	1122	17.1	1122	17.1
Equinox Framework	3.4	64301	181	56.4	182	56.0	182	56.0

表 5 不同过滤阈值下 CCN 统计信息

Table 5 CCN Information under Different Thresholds

软件名称	阈值为 15		阈值为 20		阈值为 25		阈值为 30		阈值为 35		阈值为 ∞	
	N_{CCN}	E_{CCN}	N_{CCN}	E_{CCN}	N_{CCN}	E_{CCN}	N_{CCN}	E_{CCN}	N_{CCN}	E_{CCN}	N_{CCN}	E_{CCN}
Equinox Framework	319	2130	325	2383	341	3293	346	3673	361	5054	469	40305
JDT Core	1145	9629	1192	12959	1224	15537	1255	18644	1281	21457	1635	604221
Lucene	649	3589	668	4892	718	6574	730	7159	809	10197	969	110930
PDE UI	1813	13344	1940	18041	2025	23119	2089	28461	2128	31778	2650	1062128

可以看出, 随着过滤阈值的增大, CCN 规模逐渐增大, 网络分析所需的时间和空间开销也就越大。过滤阈值为 ∞ 的情况反映了最真实完全的 CCN, 当过滤阈值为 20 以上时, 可以覆盖超过 70%的节点, 所以建议使用大于 20 的过滤阈值。

在后续的缺陷预测实验中, 构建CCN和DCN时设置的过滤阈值默认为 25。该参数变化对缺陷预测性能的影响将在 5.4.1 节实验分析。

(2) 网络嵌入算法参数

不同网络嵌入算法保持的网络结构信息不同, 目标函数不同, 涉及的参数存在差异。本节将介绍 6 种网络嵌入算法中的主要参数。

在 DeepWalk 和 Node2vec 中的可调节参数类似, number-walks 是从每个节点出发的随机游走数量, walk-length 是每个随机游走的长度。除此之外, Node2vec 可以控制随机游走的方向, 通过修改 p、q 参数即可控制到邻居的转移概率。在 LINE 中的主要调节参数有 epochs 和 order, 其中 epochs 可以控制训练的迭代次数, 而 order 可以控制算法中用到的相似度阶数。GraRep 中的 kstep 参数可以用来更改算法中的 k 步转移概率矩阵, 用来保持网络中的 k 阶相似度。GF 算法中的主要可调节参数为 epochs, 用来控制训练迭代次数。SDNE 中的 encoder-list 是一个二维向量, 其中第一维用于控制编码层的神经元数量, 第二维用于调整节点向量的维数。

本文的缺陷预测实验中网络嵌入算法借助 OpenNE[①]实现, 均使用默认的参数设置。不同参数设置对软件关联网络嵌入特征以及缺陷预测效果的影响会在 5.4.2 小节实验分析。

5.2 缺陷预测结果分析

本节展示了基于传统度量特征和基于网络嵌入特征的缺陷预测结果, 并对比了基于不同网络嵌入特征得到的缺陷预测结果。

5.2.1 CDN 的缺陷预测结果

表 6 展示了传统度量特征和 Grarep 算法得到的网络嵌入特征在 CDN 上通过随机森林分类器对 12 个数据集的缺陷预测 AUC(Area Under Curve)值。AUC 指标用于评估分类器性能, 该指标权衡了分类

① https://github.com/thunlp/OpenNE

结果的精确率和召回率。

对于每个软件, 表 6 中展示的是十次三折交叉验证得到的平均 AUC 值, 基于传统的代码度量特征的预测结果见 CM 列; 基于传统的网络度量特征的预测结果见 NM 列; 代码度量和网络度量组合的实验结果见 CM+NM 列; 基于 Grarep 算法的缺陷预测结果以及与传统度量组合后的缺陷预测结果见 NE 列、CM+NE 列和 CM+NM+NE 列。

表 6 CDN 的缺陷预测 AUC 值

Table 6 AUC Value of Defect Prediction on CDN

系统	CM	NM	CM+NM	NE	CM+NE	CM+NM +NE
Ant	0.820	0.689	0.809	0.691	**0.822**	0.816
Ivy	0.790	0.695	0.758	0.751	**0.806**	0.798
Camel	0.693	0.747	0.772	0.770	0.781	**0.803**
Lucene	0.744	0.632	0.729	0.762	0.798	**0.802**
Poi	0.867	0.849	0.872	0.861	0.894	**0.895**
Synapse	0.774	0.698	0.777	0.757	**0.802**	0.795
Tomcat	0.826	0.740	0.805	0.718	**0.834**	0.827
Velocity	0.752	0.755	0.793	0.715	0.770	**0.795**
Xalan	0.826	0.796	0.829	0.819	**0.860**	0.858
JDT Core	0.809	0.735	0.794	0.790	**0.843**	0.838
PDE UI	0.715	0.660	0.717	0.744	**0.773**	0.770
Equinox Framework	0.625	0.681	0.691	0.801	**0.799**	0.771

表 6 中加粗的字体对应每行最大值, 即每个软件使用不同度量特征进行缺陷预测时得到的最优预测结果。从表中可以看出, 最大值分布在 CM+NE、CM+NM+NE 两列。即对于传统度量来说, 结合传统度量和网络嵌入特征可以使缺陷预测达到更好的效果。

图 8 显示了在 CDN 上传统方法以及分别与 6 种网络嵌入特征结合后通过随机森林分类器得到的缺陷预测 AUC 值。由于基于代码度量的缺陷预测效果在传统方法中相对较好, 因此图 8 中用于对比的传统方法选择基于代码度量的方法。通过图 8 可以观察到, 使用了 DeepWalk、Node2vec、Grarep 后, 在绝大多数的实验对象上的预测结果均得到改善; SDNE 算法也可以在一定程度上提升预测效果; 而 GF 算法和 LINE 算法未能有效改进缺陷预测模型, 仅在 4 个实验对象上的实验效果有所提升。由此得到结论, 对于大多数实验软件, 结合基于 DeepWalk、Node2vec、Grarep 算法得到的特征可以提升在随机森林上的缺陷预测效果, 这些算法可以增加软件的网络结构信息, 从而改善缺陷预测效果。经对比计算可得, 在 CDN 上分别结合每种网络嵌入特征后的缺陷预测的性能(AUC)比仅考虑传统度量特征的预测性能, 性能提升程度的最大值是 14.6%, 平均值是 3.6%。

与上述类似, 图 9、图 10、图 11 分别显示了在 CDN 使用朴素贝叶斯、支持向量机和多层感知器进行缺陷预测的 AUC 值。可以观察到, 结合 DeepWalk、Node2vec、Grarep 嵌入特征后明显提升了在这三种分类器上的缺陷预测效果, 这与在随机森林分类器上的缺陷预测效果(见图 8)一致。通过计算可得: 结合网络嵌入特征后, 使用朴素贝叶斯性能提升最大值为 15.7%, 平均值为 3.3%; 使用支持向量机性能提升最大值为 9.6%, 平均提升为 2.5%; 使用多层感知器, 性能提升最大值为 14%, 平均值为 3.4%。使用随机森林缺陷预测的效果比朴素贝叶斯、支持向量机、多层感知器的效果要好, 性能平均高 2%、10%、9%。

分析结果 1: 在 CDN 上结合 DeepWalk、Grarep、Node2vec 特征后, 缺陷预测性能(AUC)优于仅考虑传统度量特征的预测性能, 且随机森林分类器比其他三种分类器的效果更好。

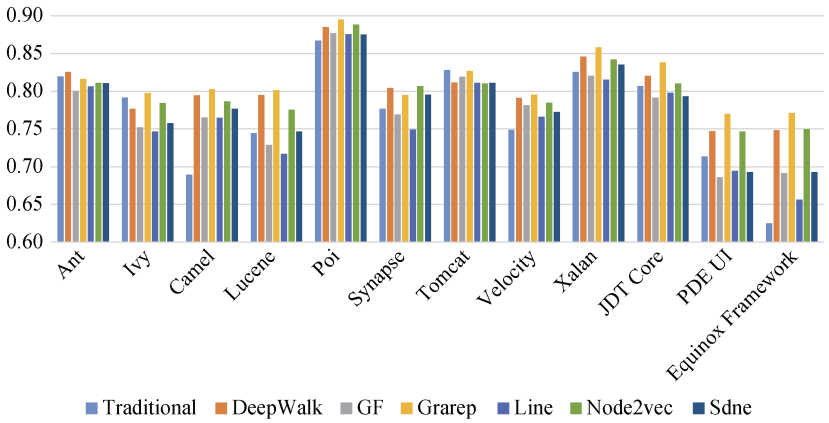


图 8 CDN 上缺陷预测的 AUC 值(使用随机森林分类)

Figure 8 AUC Value of Defect Prediction on CDN (by Random Forest)

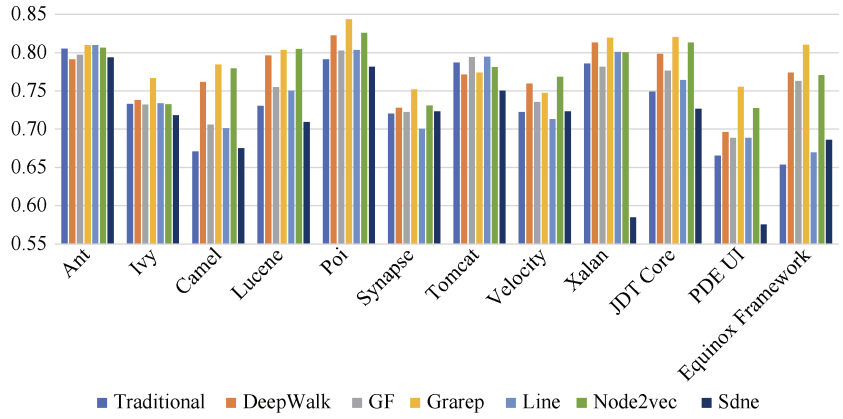


图 9　CDN 上缺陷预测的 AUC 值(使用朴素贝叶斯分类)

Figure 9　AUC Value of Defect Prediction on CDN (by Naive Bayesian)

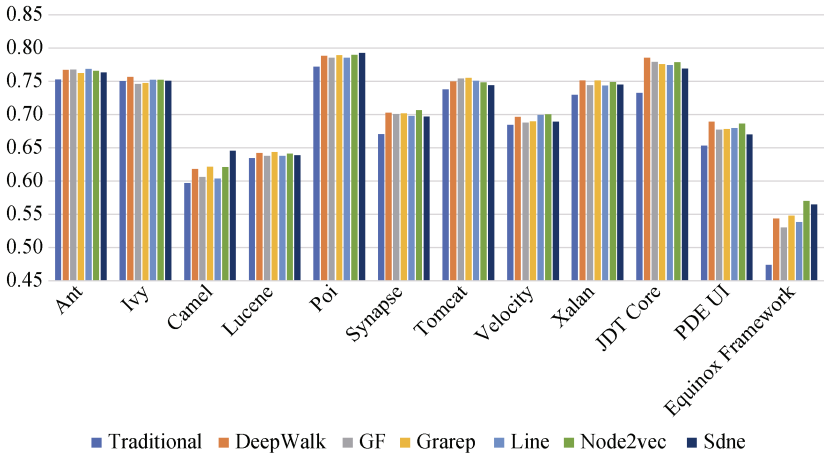


图 10　CDN 上缺陷预测的 AUC 值(使用支持向量机分类)

Figure 10　AUC Value of Defect Prediction on CDN (by Support Vector Machine)

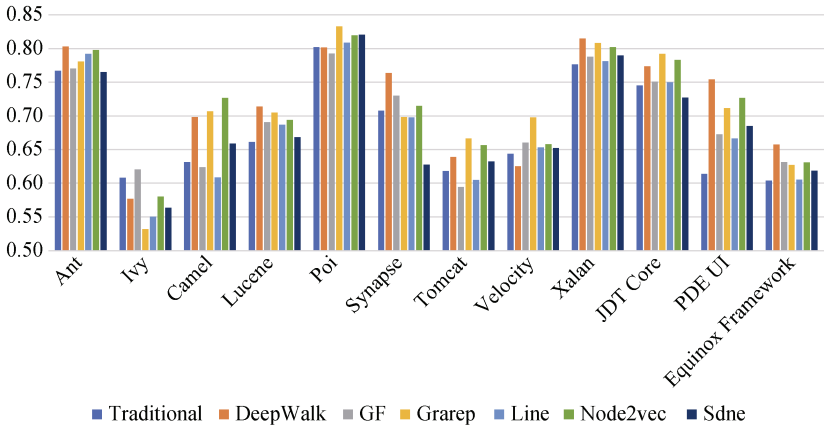


图 11　CDN 上缺陷预测的 AUC 值(使用多层感知器分类)

Figure 11　AUC Value of Defect Prediction on CDN (by Multi-Layer Perception)

5.2.2　CCN 的缺陷预测结果

表 7 展示了传统度量特征和 Grarep 算法得到的网络嵌入特征在 CCN 上通过随机森林分类得到的缺陷预测结果, 每列代表的含义和表 6 相同, CM、NM、CM+NM 列为传统方法预测结果, NE、CM+NE、CM+NM+NE 为结合网络嵌入特征的预测结果。表 7 中用加粗字体标出了每行的最大值。从加粗字体的分布可以发现, 在 CCN 中, 用 Grarep 算法得到的网络嵌入特征结合传统指标可以达到更好的缺陷预测效果。

表 7　CCN 的缺陷预测 AUC 值

Table 7　AUC Value of Defect Prediction on CCN

系统	CM	NM	CM+NM	NE	CM+NE	CM+NM +NE
Ant	0.810	0.728	0.810	0.706	**0.824**	0.818
Ivy	0.778	0.751	0.766	0.785	**0.804**	0.802
Camel	0.681	0.730	0.740	0.825	**0.837**	0.825
Lucene	0.721	0.679	0.712	0.714	**0.776**	0.752
Poi	0.864	0.684	0.863	0.759	**0.875**	0.872
Synapse	**0.774**	0.702	0.755	0.687	0.767	0.763
Tomcat	0.766	0.743	0.768	0.732	**0.777**	0.770
Velocity	0.745	0.714	0.743	0.745	**0.786**	0.779
Xalan	0.769	0.710	0.771	0.736	**0.805**	0.790
JDT Core	0.813	0.816	0.834	0.821	0.849	**0.853**
PDE UI	0.708	0.704	0.725	0.747	**0.766**	0.765
Equinox Framework	0.623	0.749	0.738	**0.794**	0.787	**0.794**

图 12 展示了在 CCN 上传统方法以及分别与 6 种网络嵌入特征结合后通过随机森林分类器得到的缺陷预测 AUC 值, 由于基于代码度量和网络度量结合的缺陷预测效果在传统方法中相对较好, 因此图 12 中对比的传统方法选择基于传统度量结合的方法。从该柱状图中观察到, LINE 和 GF 在 CCN 上仅有一半的实验对象性能得到提升, 而结合 DeepWalk、Node2vec、Grarep、SDNE 后, 缺陷预测性能都能有所改善。因此在 CCN 中使用传统度量和网络嵌入特征的结合有助于改善缺陷预测效果。经对比计算可得, 在 CCN 上结合网络嵌入特征后, 通过随机森林的缺陷预测性能相比于仅基于传统度量特征的预测方法, 性能提升程度的最大值是 17.1%, 平均值是 3.7%。

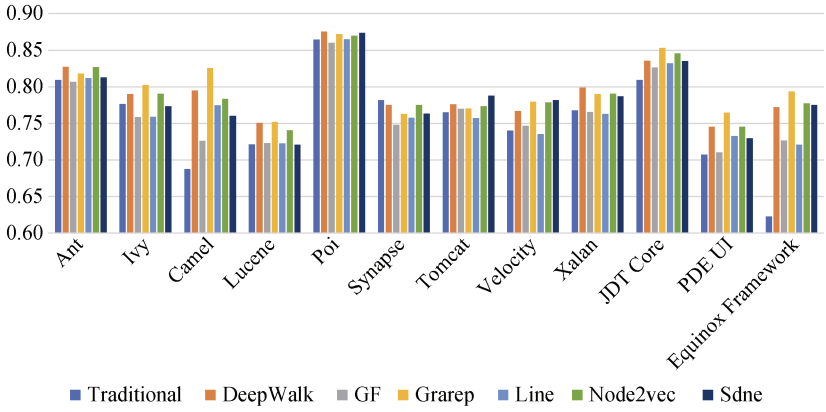


图 12　CCN 上缺陷预测的 AUC 值(使用随机森林分类)

Figure 12　AUC Value of Defect Prediction on CCN (by Random Forest)

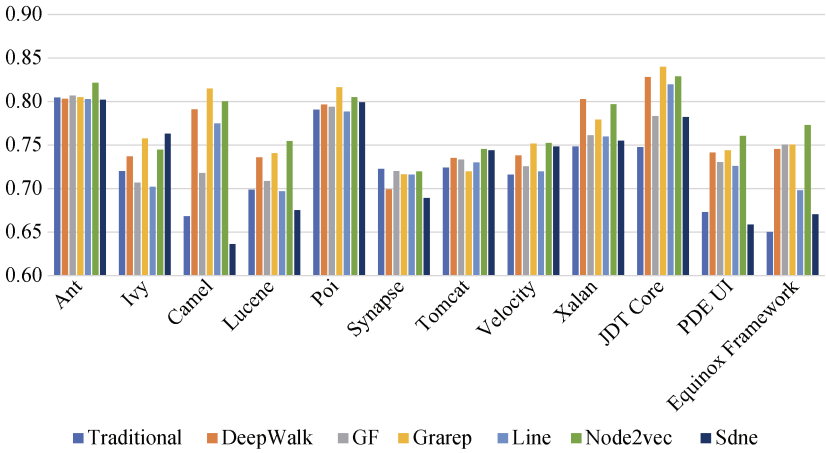


图 13　CCN 上缺陷预测的 AUC 值(使用朴素贝叶斯分类)

Figure 13　AUC Value of Defect Prediction on CCN (by Naive Bayesian)

与上述类似, 图 13、图 14、图 15 分别显示了在 CCN 使用朴素贝叶斯、支持向量机和多层感知器进行缺陷预测的 AUC 值。可以观察到, 结合 DeepWalk、Node2vec、Grarep 嵌入特征后明显提升了在这三种分类器上的缺陷预测效果, 这与在随机森林分类器上的缺陷预测效果(见图 12)一致。通过计算可得: 结合网络嵌入特征后, 使用朴素贝叶斯性能提升最大值为 12.3%, 平均值为 3.7%; 使用支持向量机性能提

升最大值为 6.5%, 平均提升为 2.2%; 使用多层感知器, 性能提升最大值为 9.7%, 平均值为 4%。使用随机森林缺陷预测的效果比朴素贝叶斯、支持向量机、多层感知器的效果要好, 性能平均高 2%、10%、10%。

分析结果 2: 在 CCN 上结合 DeepWalk、Grarep、Node2vec 特征后, 缺陷预测性能(AUC)优于仅考虑传统度量特征的预测性能, 且随机森林分类器比其他三种分类器的效果更好。

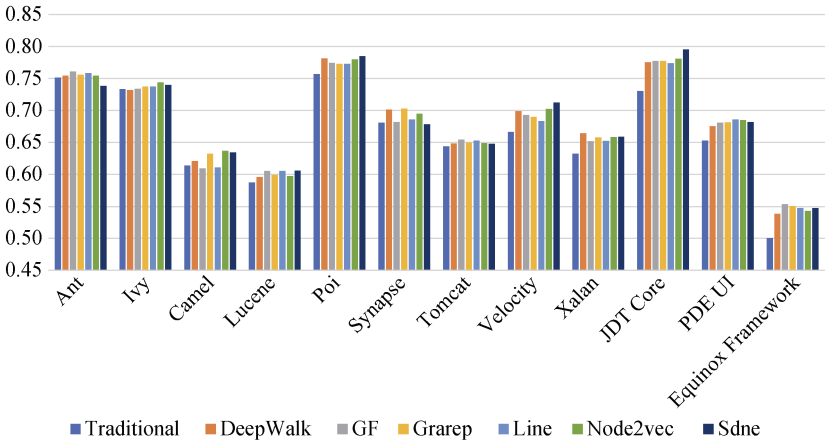


图 14 CCN 上缺陷预测的 AUC 值(使用支持向量机分类)

Figure 14 AUC Value of Defect Prediction on CCN (by Support Vector Machine)

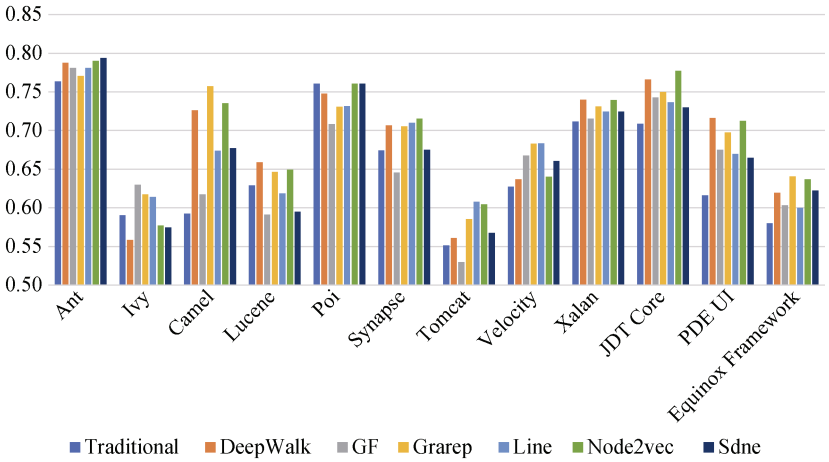


图 15 CCN 上缺陷预测的 AUC 值(使用多层感知器分类)

Figure 15 AUC Value of Defect Prediction on CCN (by Multi-Layer Perception)

5.2.3 DCN 的缺陷预测结果

表 8 展示了在 DCN 上传统度量特征和 Grarep 算法得到的网络嵌入特征通过随机森林分类器的预测结果, 每列所代表的含义和表 6 和表 7 相同, 加粗字体标出了每行的最大值。从最大值分布可以发现, 对每个实验对象来说, 缺陷预测的最好结果可能出现在传统方法中, 也可能出现在结合网络嵌入特征的方法中。也就是说, 在 DCN 中, 结合网络嵌入特征的缺陷预测结果不一定比基于传统方法的缺陷预测结果好, 没有明显的改进效果。

图 16 展示了 DCN 上传统方法以及分别与 6 种网络嵌入特征结合后通过随机森林分类器得到的缺陷预测 AUC 值。传统方法是基于代码度量的缺陷预测。从该柱状图中可以看出, 对于 12 个实验系统, 当 Grarep、SDNE 被应用于模型时, 可以提升半数实验对象的预测结果; 而其余网络嵌入算法在 DCN 上的效果不佳, 实验结果在大多数对象上均有下降。经过计算, 在 DCN 上, 结合 Grarep 算法或 SDNE 算法使得缺陷预测预测性能对比传统方法得到小幅提升, 平均提高 0.8%, 而结合其余网络嵌入算法后的缺陷预测性能均未提升。

与上述类似, 图 17、图 18、图 19 分别显示了在 DCN 使用朴素贝叶斯、支持向量机和多层感知器进行缺陷预测的 AUC 值。由图 17 可知, 当 DeepWalk、Grarep、SDNE 被应用于基于朴素贝叶斯的模型时, 可以提升多数实验对象的预测结果, 平均提升 1.1%,

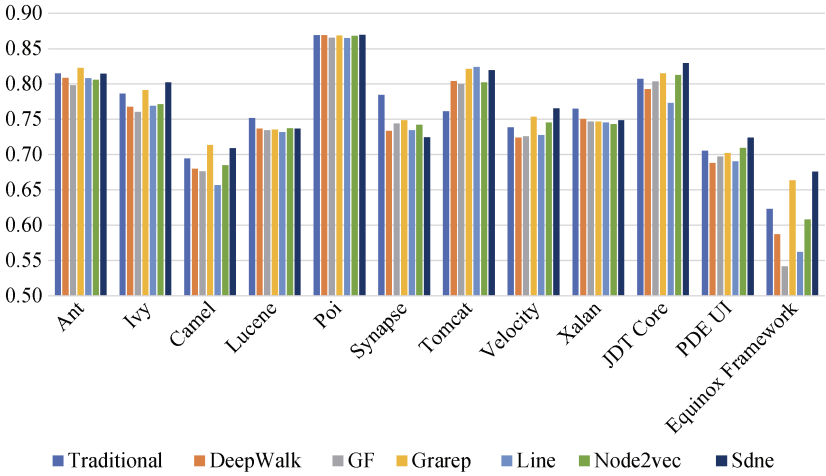


图 16　DCN 上缺陷预测的 AUC 值(使用随机森林分类)

Figure 16　AUC Value of Defect Prediction on DCN (by Random Forest)

表 8　DCN 的缺陷预测 AUC 值

Table 8　AUC Value of Defect Prediction on DCN

系统	CM	NM	CM+NM	NE	CM+NE	CM+NM +NE
Ant	0.816	0.752	0.822	0.744	**0.829**	0.822
Ivy	0.784	0.750	**0.791**	0.742	0.783	**0.791**
Camel	0.695	0.639	0.707	0.670	**0.721**	0.714
Lucene	0.752	0.690	0.722	0.602	**0.764**	0.735
Poi	0.869	0.734	0.874	0.717	**0.876**	0.869
Synapse	**0.789**	0.645	0.763	0.615	0.770	0.748
Tomcat	0.767	0.748	**0.809**	0.726	0.797	0.821
Velocity	0.741	0.676	0.729	0.682	0.749	**0.753**
Xalan	0.763	0.640	**0.767**	0.606	0.745	0.747
JDT Core	0.806	0.755	**0.824**	0.745	0.813	0.815
PDE UI	0.707	0.678	**0.712**	0.677	0.705	0.702
Equinox Framework	0.623	0.567	0.643	0.590	0.661	**0.663**

但不稳定。由图 18 可知, 分别结合 6 种网络嵌入算法在使用支持向量机分类算法后的缺陷预测结果, 在绝大多数的实验对象上的均得到小幅改善。且缺陷预测 AUC 值对比传统方法平均提高 1.8%。而通过图 19 可知, 在多层感知器上的缺陷预测模型对比传统方法没有普遍且明显的性能提升。使用随机森林缺陷预测的效果比朴素贝叶斯、支持向量机、多层感知器的效果要好, 性能平均高 2%、8%、9%。

分析结果 3: 在 DCN 上结合网络嵌入特征的缺陷预测性能(AUC)对比仅考虑传统度量特征的预测性能没有明显改善,且随机森林分类器比其他三种分类器的效果要好。

5.3　软件关联网络嵌入特征分析

5.2 节实验发现基于不同的网络嵌入特征会得到不同的缺陷预测结果。为了进一步分析原因, 本节对网络嵌入特征进行深入研究。具体来讲, 使用 k 均值聚类[81]对网络嵌入特征进行聚类, 观察聚类结果在原软件关联网络结构的分布情况, 并结合缺陷预测结果进行比较, 总结引起前述缺陷预测实验效果差异的原因。

5.3.1　网络嵌入特征的可视化

通过网络嵌入算法得到网络中节点的向量表示, 对所有向量进行 k 均值聚类, 观察节点的聚类结果在软件关联网络中的分布, 以观察软件关联网络上的网络嵌入特征特性。为了对比, 我们根据软件网络中节点的亲疏关系划分出社区(community), 每个社区内部的节点间的联系相对紧密, 各个社区之间的连接相对比较稀疏。将社区划分结果与前述基于网络嵌入的聚类结果进行对比。

图 20 展示了 Lucene 软件的类依赖网络(CDN), 网络中共 337 个点, 1306 条边。使用力引导布局来展示节点分布, 以便观察网络中节点之间的亲疏关系。网络中的节点利用 Gephi 工具①划分出社区, 连接紧密的节点位于同一社区, 每个社区内的节点分配同一种颜色。

由 k 均值聚类可以得到向量距离相近的网络节点, 我们将聚类结果表示在软件网络图中, 并借助 Gephi 工具画出, 分析节点的聚类在原软件网络中的分布。网络中的所有节点被分为 8 个簇, 不同的簇用不同的颜色表示, 两个节点如用一种颜色表示, 意

① https://gephi.org/

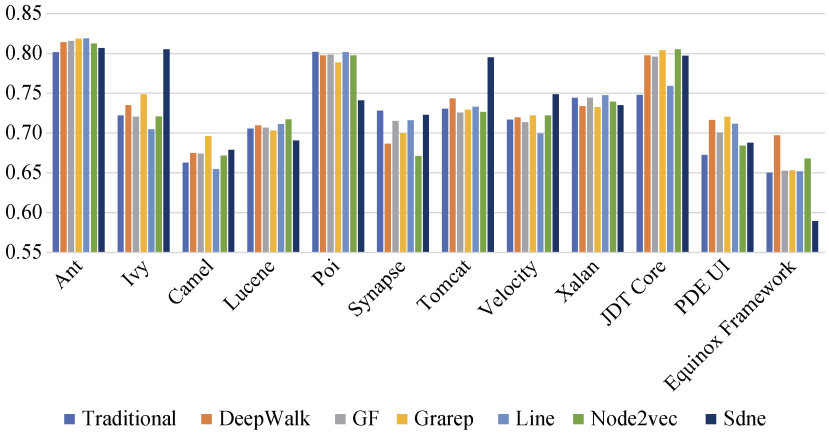


图 17　DCN 上缺陷预测的 AUC 值(使用朴素贝叶斯分类)

Figure 17　AUC Value of Defect Prediction on DCN (by Naive Bayesian)

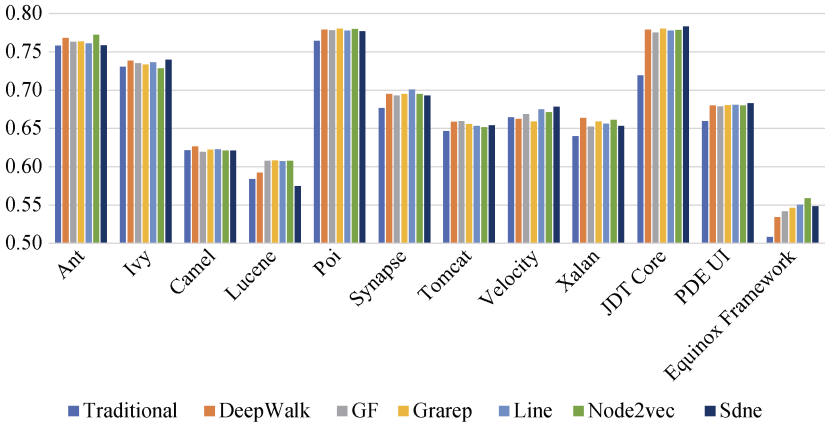


图 18　DCN 上缺陷预测的 AUC 值(使用支持向量机分类)

Figure 18　AUC Value of Defect Prediction on DCN (by Support Vector Machine)

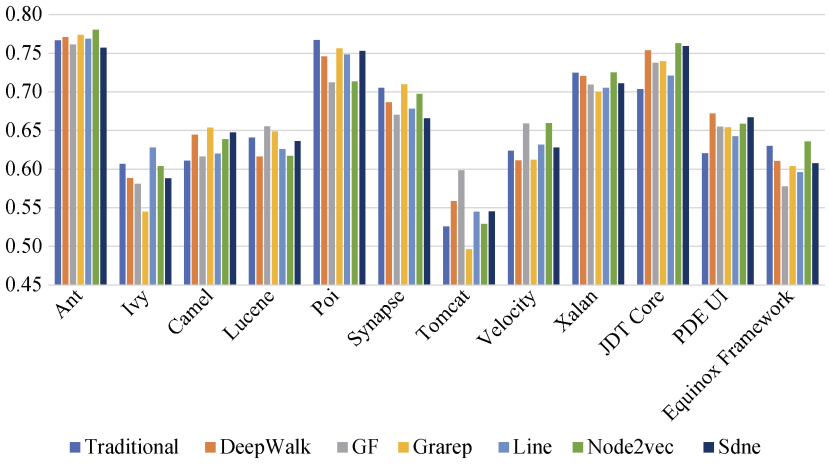


图 19　DCN 上缺陷预测的 AUC 值(使用多层感知器分类)

Figure 19　AUC Value of Defect Prediction on DCN (by Multi-Layer Perception)

味着两个节点向量的距离相近，从而被分到一个簇中，图 21 展示了 Lucene 类依赖网络的节点分别基于 6 种网络嵌入特征的聚类结果。

从图 21 可以发现，不同网络嵌入的节点的聚类结果不一样。分析图中不同颜色节点的分布，例如蓝色的节点，经过 DeepWalk、Node2vec 和 Grarep 算法的聚类效果接近，这 3 种网络嵌入算法均将原软件网络结构中连接紧密的节点聚为一类，同一颜色的节点都相距很近。而 LINE 和 GF 算法将原软件网络中相距很远的节点映射到了相近的向量空间中，从

而聚类为一簇, 聚类结果与原软件网络没有明显的相关性。而基于 SDNE 算法的节点聚类结果图将网络中 78%的节点聚为一类(橙色节点)。其余未展示出的 11 个软件也有同样类似的聚类结果。

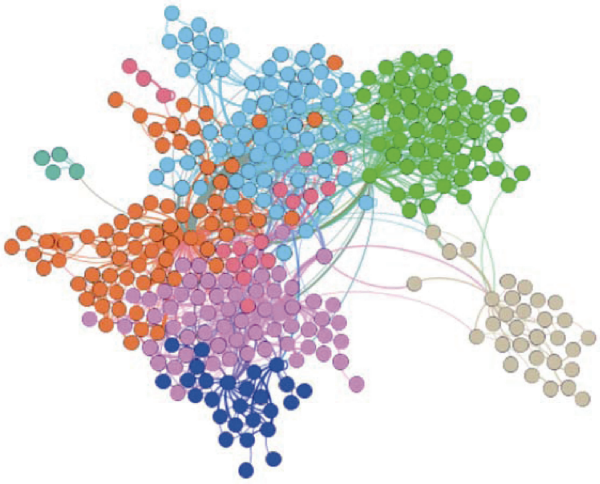


图 20 Lucene 软件类依赖网络(CDN)
Figure 20 Class Dependency Network(CDN) of Lucene

分析结果 4: 由上述可视化结果初步观察到, DeepWalk、Grarep 和 Node2vec 算法比 LINE 和 GF 算法更擅长学习网络的同质性[16], 即倾向于将原软件网络中连接紧密的节点在向量空间中的距离也表示的相近。而 LINE 和 GF 算法却可能将网络中连接疏松的节点映射为在向量空间中相近的节点, 与原软件网络没有明显的相关性。

5.3.2 网络嵌入特征保持的网络结构分析

为了进一步分析 6 种网络嵌入算法学习网络同质性的能力, 本节用 *MoJoFM* 值[85]衡量基于网络嵌入的聚类结果与社区结构的相似程度(例如图 11 和图 12 之间的相似程度)。*MoJoFM* 通过计算从一个结构转换到另一个结构所需的操作来度量两个结构之间的相似程度。较高的 *MoJoFM* 值意味着两个结构更相似。*MoJoFM* 值介于 0 和 1 之间, 0 表示最不相似, 1 表示完全一样。*MoJoFM* 的公式如下:

$$MoJoFM(OA_i, OA_j) = 1 - \frac{mno(OA_i, OA_j)}{\max(mno(\forall OA_i, OA_j))}$$

其中, $mno(OA_i, OA_j)$ 表示从一个结构OA_i转变成另一个结构OA_j所需的Move操作和Join操作的数目。

表 9 对网络中划分的社区与 6 种网络嵌入特征的聚类结果的相似程度进行对比, 例如 Lucene 行 DeepWalk 列, *MoJoFM* 值为 47.98%, 说明 Lucene 的类依赖网络的社区结构与基于 DeepWalk 算法得到的聚类结果之间的相似程度为 47.98%。*MoJoFM* 值越大, 则说明聚类结果与社区的划分越相似, 网络嵌入算法能更好地保持网络中连接紧密的节点的结构特征, 对网络同质性的学习能力越强。

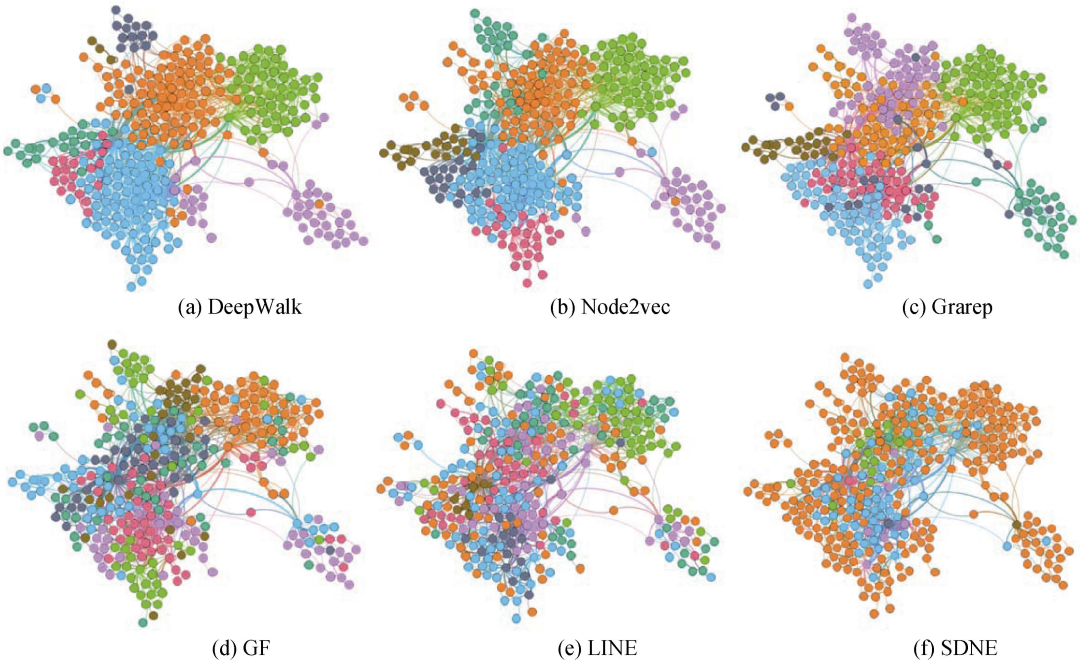


图 21 Lucene 类依赖网络的节点的聚类结果
Figure 21 Clusters of Class Dependency Network for Lucene

表 9 Lucene 的社区与聚类之间的 *MoJoFM* 值

Table 9 *MoJoFM* between Communities and Clusters for Lucene (%)

	DeepWalk	GF	Grarep	LINE	Node2vec	SDNE
Ant	67.60	32.96	60.45	27.77	74.19	66.48
Ivy	45.05	20.72	38.44	14.11	45.35	56.80
Camel	48.76	23.25	39.39	26.75	53.61	40.97
Lucene	47.98	28.66	52.65	23.05	52.96	63.61
Poi	63.53	30.26	62.88	41.13	50.35	53.02

从表 9 可以发现, 对软件关联网络划分的社区与不同网络嵌入特征的聚类结果之间的 *MoJoFM* 值存在差别, 其中基于 DeepWalk、Grarep、Node2vec 算法的结果明显高于基于 GF、LINE 的结果。也就是说, 基于 DeepWalk、Grarep、 Node2vec 算法的聚类结果, 与网络的社区之间的相似程度更大, 可以使网络中紧密相连的节点映射到相近的向量空间。

分析结果 5:该实验的定量分析结果验证了 5.3.1 节的可视化观察, 即: DeepWalk、Grarep 和 Node2vec 算法比 LINE 和 GF 算法更擅长学习网络的同质性。

5.3.3 网络嵌入特征的缺陷预测性能分析

本节中用 *MoJoFM* 值对不同网络嵌入特征的聚类结果之间的相似性进行衡量, 并观察聚类结果的相似性与缺陷预测结果差值之间的关系。本节首先分析不同网络嵌入特征保持的软件网络结构信息之间的相似程度。用 *MoJoFM* 值衡量 5.3.1 节获得的 6 种网络嵌入特征的聚类结果之间的相似程度, *MoJoFM* 值越大, 说明聚类结果越相似, 也就是网络嵌入特征保持的网络结构信息越相似。然后分析聚类结果相似性与相应缺陷预测性能差异之间的关系, 表 10 和表 11 展示 Lucene 软件基于不同网络嵌入的聚类结果之间的相似性与缺陷预测 AUC 差值。

对照表 10 和表 11 中的值可以发现, 对于 Lucene 软件, 两种网络嵌入特征的聚类结果之间相似度(*MoJoFM*)大于 50%时, 则基于这两种网络嵌入的缺陷预测 AUC 差值不超过 0.03。

表 10 Lucene 软件的网络嵌入聚类结果之间的 *MoJoFM* 值

Table 10 *MoJoFM* between Different Network Embedding Clusters for Lucene (%)

	DeepWalk	GF	Grarep	LINE	Node2vec	SDNE
DeepWalk	100					
GF	35.20	100				
Grarep	**70.22**	33.06	100			
LINE	26.13	18.72	28.27	100		
Node2vec	**67.56**	36.54	**61.97**	28.27	100	
SDNE	39.55	31.95	41.10	36.65	38.43	100

表 11 Lucene 软件基于不同网络嵌入的缺陷预测 AUC 差值

Table 11 AUC Difference Based on Different Network Embedding Algorithms for Lucene

	DeepWalk	GF	Grarep	LINE	Node2vec	SDNE
DeepWalk	0					
GF	0.11	0				
Grarep	**0.03**	0.14	0			
LINE	0.10	0.01	0.13	0		
Node2vec	**0.01**	0.10	**0.03**	0.09	0	
SDNE	0.02	0.09	0.05	0.08	0.02	0

对数据集中 12 个软件进行同样分析后发现类似的结果: 对某一软件, 用 *MoJoFM* 值度量任意两种网络嵌入特征的聚类结果之间相似度, 相似度大于 50%则缺陷预测结果相近, 一般情况下缺陷预测 AUC 值之间相差小于 0.05。并且由前述可知, DeepWalk、Grarep 和 Node2vec 算法均擅长学习网络的同质性, 它们的缺陷预测性能良好且相近。

分析结果 6: 当两种网络嵌入算法保持的软件关联网络的结构特征相似时, 其缺陷预测效果也相似; 且当网络嵌入能保持网络结构的同质性时, 则缺陷预测性能良好。

5.4 参数分析

本节分析不同的参数设置对缺陷预测带来的影响。5.4.1 节展示了预处理时过滤阈值的选取对缺陷预测结果的影响。5.4.2 节展示了网络嵌入算法在不同参数设置下的缺陷预测结果。

5.4.1 过滤阈值参数的影响分析

在数据集预处理中提到, 在构建 CCN 和 DCN 时需要找到最适用于缺陷预测模型的过滤阈值。图 22 展示了在不同过滤阈值下, 在 CCN 使用传统度量的缺陷预测 AUC 值。

如图 22 所示, 预测效果常在 10~25 之间上升, 在阈值为 25 的时候达到最佳, 在 30 和 35 时又有点回落。根据表 5, 阈值为 25 时的网络已经包含了超过 70%的真实节点信息。注意到在 Equinox Framework 上的实验阈值为 10 时的 AUC 值超过了阈值为

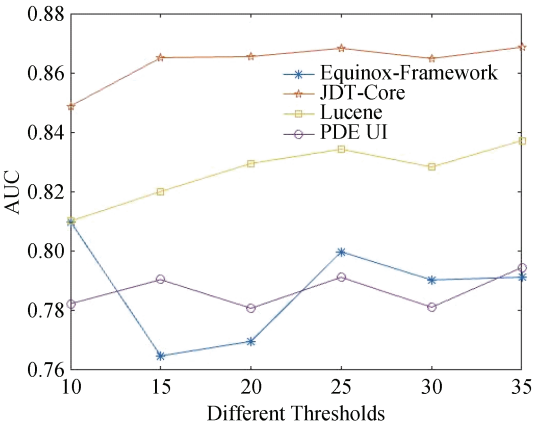


图 22　不同过滤阈值下基于 CCN 的缺陷预测 AUC 值

Figure 22　AUC Value of Defect Prediction under Different Thesholds on CCN

25 的时候, 但这只是个例, 不具普遍性; 且阈值为 10 包含的节点信息过少, 无法准确地反映真实网络的情况。因而, 本文在前文缺陷预测实验中过滤阈值参数也是设置为 25 (见 5.1.2 节)。

分析结果 7: 将 CCN 和 DCN 用于缺陷预测时, 过滤阈值参数设置为 25 可使得缺陷预测性能得到最佳。

5.4.2　网络嵌入算法参数的影响分析

本节对网络嵌入算法的参数进行修改, 得到基于不同参数的网络嵌入的缺陷预测结果。每个网络嵌入算法选择一个主要影响参数进行修改, 表 12 和表 13 展示了 5 个软件在 CDN 上的基于不同参数的网络嵌入的缺陷预测结果。分析可得:

DeepWalk 算法的步长参数从 80 改为 40 后, 缺陷预测效果没有明显差异。

GF 算法的 epochs 参数从 5 调整为 100 后, 缺陷预测效果在所有软件中均有大幅提升。

Grarep 算法中的 kstep 参数不同时, 可以得到不同阶次的全局信息, 将 kstep 从 4 调整为 8 后, 得到的缺陷预测结果并没有大的变动, 说明调整参数前后, 算法保持的网络结构信息相似。

LINE 算法中的 epochs 参数从 5 调整为 100 后, 与 GF 算法类似, 缺陷预测效果在所有软件中均有大幅提升。

Node2vec 算法的 p、q 参数用于控制随机游走的方向。当 p=1、q=0.5 时, 算法对比 p=1、q=1 更擅长学习网络的同质性。并且从缺陷预测结果可以观察到, 当算法参数选择 p=1、q=0.5 时, 拥有更好的缺陷预测效果, 即连接紧密的节点在低维向量空间中的表示结果也相近时, 可以获得更好的缺陷预测效果, 这与 5.3 节中得到的结论是一致的。

SDNE 算法修改 encoder-list 参数的第一项, 将编码器的神经元个数从 1000 改为 500, 缺陷预测效果变化较小, 说明调整该参数前后, 算法保持的网络结构信息相似。

分析结果 8: 网络嵌入算法参数发生变化时, 相应的缺陷预测效果会受到不同程度的影响。特别是, 缺陷预测性能对 Node2vec、LINE、GF 算法的参数更为敏感。

表 12　基于网络嵌入的缺陷预测 AUC 值

Table 12　AUC Value of Defect Prediction Based on Network Embedding

	DeepWalk_ walklength=80	GF_ epochs=5	Grarep_ kstep=4	LINE_ epochs=5	Node2vec_ p=1_q=1	SDNE_ encoder-list=[1000,32]
Ant	0.70	0.53	0.69	0.59	0.66	0.70
Ivy	0.63	0.53	0.75	0.46	0.59	0.67
Camel	0.74	0.58	0.77	0.64	0.71	0.69
Lucene	0.73	0.62	0.76	0.64	0.73	0.71
Poi	0.84	0.69	0.86	0.74	0.83	0.83

表 13　基于网络嵌入的缺陷预测 AUC 值

Table 13　AUC Value of Defect Prediction Based on Network Embedding

	DeepWalk _ walklength=40	GF_ epochs=100	Grarep_ kstep=8	LINE_ epochs=100	Node2vec_ p=1_q=0.5	SDNE_ encoder-list=[500,32]
Ant	0.68	0.60	0.71	0.67	0.67	0.69
Ivy	0.64	0.58	0.75	0.60	0.65	0.67
Camel	0.73	0.66	0.76	0.71	0.72	0.73
Lucene	0.75	0.70	0.77	0.72	0.75	0.74
Poi	0.83	0.81	0.86	0.84	0.82	0.84

5.5 实验小结

本章研究了基于不同网络嵌入特征的缺陷预测结果。首先展示了实验的数据集, 对实验软件构建了3 种网络——CDN、CCN 和 DCN, 将传统度量特征和 6 种网络嵌入特征与对应缺陷标签输入至 4 种分类算法中进行训练并测试, 并用 AUC 值衡量缺陷预测效果, 总结见表 14。然后分析了网络嵌入保持的网络结构特征以及缺陷预测性能差异的原因, 最后分析了不同参数设置下的网络嵌入特征对缺陷预测性能的影响。

表 14 面向缺陷预测的网络嵌入算法预测效果总结
Table 14 Summary for Defect Prediction Based on Network Embedding

算法名称	类依赖网络 CDN	文件耦合网络 CCN	开发者贡献网络 DCN
DeepWalk	√较好	√较好	×较差
GF	×较差	×较差	×较差
Grarep	√较好	√较好	-一般
LINE	×较差	×较差	×较差
Node2vec	√较好	√较好	×较差
SDNE	-一般	√较好	-一般

总体而言: 1) 除了开发者贡献网络 DCN 之外, 在类依赖网络CDN和文件耦合网络CCN上, 在传统的度量特征上结合网络嵌入特征后, 缺陷预测性能得到显著提升; 2)使用随机分类器的效果要比朴素贝叶斯、支持向量机、多层感知器的效果要好; 3) DeepWalk、Grarep 和 Node2vec 这 3 种网络嵌入算法保持的网络结构特性类似, 缺陷预测效果相近; 4) 与 LINE 和 GF 算法不同, DeepWalk、Grarep 和 Node2vec 算法更擅长学习网络的同质性, 因而在相同软件关联网络上的缺陷预测效果更好; 5) 网络嵌入算法在不同的参数设置下保持的网络结构特性存在差异, 缺陷预测性能对网络嵌入算法参数敏感, 实验中应根据观察和经验选择合适的参数。

6 总结和未来工作

本文主要研究面向软件缺陷预测的网络嵌入特征, 设计了一种基于软件关联网络嵌入的缺陷预测方法, 全面分析了不同软件关联网络、不同网络嵌入算法、不同参数设置下的网络嵌入特征及其缺陷预测性能。通过实验分析, 主要发现类依赖网络和文件耦合网络比开发者贡献网络更能显著提升缺陷预测效果; DeepWalk、Grarep 和 Node2vec 算法比 LINE 和 GF 算法更擅长保持网络结构的同质性, 因而这 3 种算法下的缺陷预测效果最好。此外, 缺陷预测对网络嵌入算法参数设置敏感。总之, 本文得出的实验结论有助于指导缺陷预测活动中如何选择软件关联网络和网络嵌入算法以获取较好的性能。

未来将进一步完善本文工作, 包括: 1)本文构建的类依赖网络是针对面向对象特性, 使用的实验对象是 Java 实现的软件系统。在应用于其他编程语言如 C/C++、Python 等实现的系统时, 需要扩展类依赖网络, 以更准确的描述软件网络结构, 我们未来将对这一部分进行探索。2)本文探究了对相同软件使用不同网络嵌入算法得到的缺陷预测效果的差异性原因, 也观察到不同软件上的缺陷预测效果存在一定差异。因此未来将分析不同软件系统上预测效果差异的原因。3)除了本文使用的 AUC 外, 还有其他评价分类性能的指标, 例如对于不均衡数据集的评估非常有效的 MCC(Matthews Correlation Coefficient)指标。未来将使用更多的相关指标进一步验证本文缺陷预测效果。

致 谢 在此向给予指导的老师、提供帮助的同学和给本文提出建议的评审专家表示感谢。并且感谢国家重点研发计划资助项目(No.2018YFB1004500), 国家自然科学基金(No.61632015, No.61772408, No.U1766215, No.61721002, No.61833015, No.62002280, No.61902306, No.61602369), 国网陕西省电力公司科技项目(No.5226SX1800FC), 教育部创新团队(No.IRT_17R86)和中国工程科技知识中心项目, 中国博士后资助项目(No.2019TQ0251, No.2020M673439)的资助。

参考文献

[1] Wang Q, Wu S J, Li M S. Software Defect Prediction[J]. *Journal of Software*, 2008, 19(7): 1565-1580.
(王青, 伍书剑, 李明树. 软件缺陷预测技术[J]. *软件学报*, 2008, 19(7): 1565-1580.)

[2] Martino S, Ferrucci F, Gravino C, et al. A genetic algorithm to configure support vector machines for predicting fault-prone components[C]. *International conference on product focused software process improvement*. 2011: 247-261.

[3] Nguyen T H D, Adams B, Hassan A E. Studying the Impact of Dependency Network Measures on Software Quality[C]. *2010 IEEE International Conference on Software Maintenance*, 2010: 1-10.

[4] Zimmermann T, Nagappan N. Predicting Defects Using Network Analysis on Dependency Graphs[C]. *2008 ACM/IEEE 30th Inter-*

national Conference on Software Engineering, 2008: 531-540.

[5] NASA IV&V Facility Metrics Data Program. http://mdp.ivv.nasa.gov, 2010.

[6] M. H. Halstead. Elements of Software Science[M]. (Operating and Programming Systems Series)," Elsevier Science Inc, 1977.

[7] McCabe T J. A Complexity Measure[J]. *IEEE Transactions on Software Engineering*, 1976, SE-2(4): 308-320.

[8] Chidamber S R, Kemerer C F. A Metrics Suite for Object Oriented Design[J]. *IEEE Transactions on Software Engineering*, 1994, 20(6): 476-493.

[9] Chen L, Ma W, Zhou Y M, et al. Empirical Analysis of Network Measures for Predicting High Severity Software Faults[J]. *Science China Information Sciences*, 2016, 59(12): 1-18.

[10] Ma W, Chen L, Yang Y B, et al. Empirical Analysis of Network Measures for Effort-Aware Fault-Proneness Prediction[J]. *Information and Software Technology*, 2016, 69: 50-70.

[11] Jin W X, Cai Y F, Kazman R, et al. ENRE: A Tool Framework for Extensible eNtity Relation Extraction[C]. *2019 IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings*, 2019: 67-70.

[12] Goyal P, Ferrara E. Graph Embedding Techniques, Applications, and Performance: A Survey[J]. *Knowledge-Based Systems*, 2018, 151: 78-94.

[13] Cao S S, Lu W, Xu Q K. GraRep: Learning Graph Representations with Global Structural Information[C]. *CIKM '15: Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*. 2015: 891-900.

[14] Ahmed A, Shervashidze N, Narayanamurthy S, et al. Distributed Large-Scale Natural Graph Factorization[C]. *The 22nd international conference on World Wide Web - WWW '13*, 2013: 37-48.

[15] Perozzi B, Al-Rfou R, Skiena S. DeepWalk: Online Learning of Social Representations[C]. *KDD '14: Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. 2014: 701-710.

[16] Grover A, Leskovec J. Node2vec: Scalable Feature Learning for Networks[C]. The 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2016: 855-864.

[17] Wang D X, Cui P, Zhu W W. Structural Deep Network Embedding[C]. *The 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2016: 1225-1234.

[18] Feder T, Motwani R. Clique Partitions, Graph Compression and Speeding-up Algorithms[J]. *Journal of Computer and System Sciences*, 1995, 51(2): 261-272.

[19] Tian Y Y, Hankins R A, Patel J M. Efficient Aggregation for Graph Summarization[C]. *The 2008 ACM SIGMOD international conference on Management of data - SIGMOD '08*, 2008: 567-580.

[20] Navlakha S, Rastogi R, Shrivastava N. Graph Summarization with Bounded Error[C]. *The 2008 ACM SIGMOD international conference on Management of data - SIGMOD '08*, 2008: 419-432.

[21] D. Jungnickel and T. Schade. Graphs, networks and algorithms[M]. Springer, 2005.

[22] Battista G D, Eades P, Tamassia R, et al. Algorithms for Drawing Graphs: An Annotated Bibliography[J]. *Computational Geometry*, 1994, 4(5): 235-282.

[23] Eades P, Lin X M. How to Draw a Directed Graph[C]. *1989 IEEE Workshop on Visual Languages*, 1989: 13-17.

[24] White S, Smyth P. A Spectral Clustering Approach to Finding Communities in Graphs[C]. *The 2005 SIAM International Conference on Data Mining*, 2005: 274-285.

[25] Liben-Nowell D, Kleinberg J. The Link-Prediction Problem for Social Networks[J]. *Journal of the American Society for Information Science and Technology*, 2007, 58(7): 1019-1031.

[26] Lü L, Zhou T. Link Prediction in Complex Networks: A Survey[J]. *Physica A: Statistical Mechanics and Its Applications*, 2011, 390(6): 1150-1170.

[27] Hasan M A, Zaki M J. A Survey of Link Prediction in Social Networks[M]. Social Network Data Analytics. Boston, MA: Springer US, 2011: 243-275.

[28] Bhagat S, Cormode G, Muthukrishnan S. Node Classification in Social Networks[M]. Social Network Data Analytics. Boston, MA: Springer US, 2011: 115-148.

[29] Tang J, Qu M, Wang M Z, et al. LINE: Large-Scale Information Network Embedding[C]. *The 24th International Conference on World Wide Web*, 2015: 1067-1077.

[30] Chen X, Gu Q, Liu W S, et al. Survey of Static Software Defect Prediction[J]. *Journal of Software*, 2016, 27(1): 1-25.
(陈翔, 顾庆, 刘望舒, 等. 静态软件缺陷预测方法研究[J]. *软件学报*, 2016, 27(1): 1-25.)

[31] Elish K O, Elish M O. Predicting Defect-Prone Software Modules Using Support Vector Machines[J]. *Journal of Systems and Software*, 2008, 81(5): 649-660.

[32] Catal C, Diri B N. Investigating the Effect of Dataset Size, Metrics Sets, and Feature Selection Techniques on Software Fault Prediction Problem[J]. *Information Sciences*, 2009, 179(8): 1040-1058.

[33] Kim S, Zimmermann T, Whitehead E J Jr, et al. Predicting Faults from Cached History[C]. *29th International Conference on Software Engineering (ICSE'07)*, 2007: 489-498.

[34] Rahman F, Posnett D, Hindle A, et al. BugCache for Inspections: Hit or Miss?[C]. *The 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering - SIGSOFT/FSE '11*, 2011: 322-331.

[35] Rahman F, Posnett D, Herraiz I, et al. Sample Size Vs. Bias in Defect Prediction[C]. *The 2013 9th Joint Meeting on Foundations of Software Engineering - ESEC/FSE 2013*, 2013: 147-157.

[36] Shatnawi R. Improving Software Fault-Prediction for Imbalanced Data[C]. *2012 International Conference on Innovations in Information Technology*, 2012: 54-59.

[37] Pelayo L, Dick S. Applying Novel Resampling Strategies to Software Defect Prediction[C]. *NAFIPS 2007 - 2007 Annual Meeting of the North American Fuzzy Information Processing Society*, 2007: 69-72.

[38] Liu Y B, Li Y H, Guo J B, et al. Connecting Software Metrics across Versions to Predict Defects[C]. *2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering*, 2018: 232-243.

[39] Herzig K, Just S, Rau A, et al. Predicting Defects Using Change Genealogies[C]. *2013 IEEE 24th International Symposium on Software Reliability Engineering*, 2013: 118-127.

[40] Nagappan N, Ball T. Use of Relative Code Churn Measures to Predict System Defect Density[C]. *27th International Conference on Software Engineering, 2005. ICSE 2005*, 2005: 284-292.

[41] Moser R, Pedrycz W, Succi G. A Comparative Analysis of the Efficiency of Change Metrics and Static Code Attributes for Defect Prediction[C]. *2008 ACM/IEEE 30th International Conference on Software Engineering*, 2008: 181-190.

[42] Hassan A E. Predicting Faults Using the Complexity of Code Changes[C]. *2009 IEEE 31st International Conference on Software Engineering*, 2009: 78-88.

[43] Pinzger M, Nagappan N, Murphy B. Can Developer-Module Networks Predict Failures[C]. *The 16th ACM SIGSOFT International Symposium on Foundations of software engineering - SIGSOFT '08/FSE-16*, 2008: 2-12.

[44] Meneely A, Williams L, Snipes W, et al. Predicting Failures with Developer Networks and Social Network Analysis[C]. *The 16th ACM SIGSOFT International Symposium on Foundations of software engineering - SIGSOFT '08/FSE-16*, 2008: 13-23.

[45] Jiang T, Tan L, Kim S. Personalized Defect Prediction[C]. *2013 28th IEEE/ACM International Conference on Automated Software Engineering* , 2013: 279-289.

[46] Bird C, Nagappan N, Gall H, et al. Putting it all Together: Using Socio-Technical Networks to Predict Failures[C]. *2009 20th International Symposium on Software Reliability Engineering*, 2009: 109-119.

[47] Nagappan N, Murphy B, Basili V. The Influence of Organizational Structure on Software Quality: An Empirical Case Study[C]. *The 13th international conference on Software engineering - ICSE '08*, 2008: 521-530.

[48] Mockus A. Organizational Volatility and Its Effects on Software Defects[C]. *The eighteenth ACM SIGSOFT international symposium on Foundations of software engineering - FSE '10*, 2010: 117-126.

[49] Bird C, Nagappan N, Devanbu P, et al. Does Distributed Development Affect Software Quality? an Empirical Case Study of Windows Vista[C]. *2009 IEEE 31st International Conference on Software Engineering*, 2009: 518-528.

[50] Barabasi, Albert. Emergence of Scaling in Random Networks[J]. *Science*, 1999, 286(5439): 509-512.

[51] Watts D J, Strogatz S H. Collective Dynamics of 'Small-World' Networks[J]. *Nature*, 1998, 393(6684): 440-442.

[52] Myers C R. Software Systems as Complex Networks: Structure, Function, and Evolvability of Software Collaboration Graphs[J]. *Physical Review E, Statistical, Nonlinear, and Soft Matter Physics*, 2003, 68(4 pt 2): 046116.

[53] Concas G, Marchesi M, Pinna S, et al. Power-Laws in a Large Object-Oriented Software System[J]. *IEEE Transactions on Software Engineering*, 2007, 33(10): 687-708.

[54] Louridas P, Spinellis D, Vlachos V. Power Laws in Software[J]. *ACM Transactions on Software Engineering and Methodology*, 2008, 18(1): 1-26.

[55] Li H, Zhao H, Cai W, et al. A Modular Attachment Mechanism for Software Network Evolution[J]. *Physica A: Statistical Mechanics and Its Applications*, 2013, 392(9): 2025-2037.

[56] Turnu I, Concas G, Marchesi M, et al. A Modified Yule Process to Model the Evolution of some Object-Oriented System Properties[J]. *Information Sciences*, 2011, 181(4): 883-902.

[57] Bhattacharya P, Iliofotou M, Neamtiu I, et al. Graph-Based Analysis and Prediction for Software Evolution[C]. *2012 34th International Conference on Software Engineering*, 2012: 419-429.

[58] Cai K Y, Yin B B. Software Execution Processes as an Evolving Complex Network[J]. *Information Sciences*, 2009, 179(12): 1903-1928.

[59] Concas G, Marchesi M, Monni C, et al. Software Quality and Community Structure in Java Software Networks[J]. *International Journal of Software Engineering and Knowledge Engineering*, 2017, 27(7): 1063-1096.

[60] Pan W F, Chai C L. Measuring Software Stability Based on Complex Networks in Software[J]. *Cluster Computing*, 2019, 22(2): 2589-2598.

[61] Fan M, Liu J, Luo X P, et al. Android Malware Familial Classification and Representative Sample Selection via Frequent Subgraph Analysis[J]. *IEEE Transactions on Information Forensics and Security*, 2018, 13(8): 1890-1905.

[62] Fan M, Wei W Y, Xie X F, et al. Can we Trust your Explanations? Sanity Checks for Interpreters in Android Malware Analysis[J]. *IEEE Transactions on Information Forensics and Security*, 2021, 16: 838-853.

[63] Fan M, Liu J, Wang W, et al. DAPASA: Detecting Android Piggybacked Apps through Sensitive Subgraph Analysis[J]. *IEEE*

Transactions on Information Forensics and Security, 2017, 12(8): 1772-1785.

[64] Šubelj L, Bajec M. Community Structure of Complex Software Systems: Analysis and Applications[J]. *Physica A: Statistical Mechanics and Its Applications*, 2011, 390(16): 2968-2975.

[65] Qu Y, Guan X H, Zheng Q H, et al. Exploring Community Structure of Software Call Graph and Its Applications in Class Cohesion Measurement[J]. *Journal of Systems and Software*, 2015, 108: 193-210.

[66] Pan W F, Li B, Liu J, et al. Analyzing the Structure of Java Software Systems by Weighted K-Core Decomposition[J]. *Future Generation Computer Systems*, 2018, 83: 431-444.

[67] Belkin M, Niyogi P. Laplacian eigenmaps and spectral techniques for embedding and clustering[C]. *Advances in neural information processing systems*. 2002: 585-591.

[68] Roweis S T. Nonlinear Dimensionality Reduction by Locally Linear Embedding[J]. *Science*, 2000, 290(5500): 2323-2326.

[69] Fan M, Luo X P, Liu J, et al. Graph Embedding Based Familial Analysis of Android Malware Using Unsupervised Learning[C]. *2019 IEEE/ACM 41st International Conference on Software Engineering*, 2019: 771-782.

[70] Ribeiro L F R, Saverese P H P, Figueiredo D R. Struc2vec: Learning Node Representations from Structural Identity[C]. *The 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2017: 385-394.

[71] Qu Y, Liu T, Chi J L, et al. Node2defect: Using Network Embedding to Improve Software Defect Prediction[C]. *The 33rd ACM/IEEE International Conference on Automated Software Engineering*, 2018: 844-849.

[72] Liu C B, Zheng W, Fan X, et al. Hybrid Defect Prediction Model Based on Network Representation Learning[J]. *Journal of Computer Applications*, 2019, 39(12): 3633-3638.
(刘成斌, 郑巍, 樊鑫, 等. 基于网络表征学习的混合缺陷预测模型[J]. *计算机应用*, 2019, 39(12): 3633-3638.)

[73] Ling C Y, Zou Y Z, Lin Z Q, et al. Approach to Searching Software Source Code with Graph Embedding[J]. *Journal of Software*, 2019, 30(5): 1481-1497.
(凌春阳, 邹艳珍, 林泽琦, 等. 基于图嵌入的软件项目源代码检索方法[J]. *软件学报*, 2019, 30(5): 1481-1497.)

[74] Kurimoto S, Hayase Y, Yonai H, et al. Class Name Recommendation Based on Graph Embedding of Program Elements[C]. *2019 26th Asia-Pacific Software Engineering Conference*, 2019: 498-505.

[75] Yonai H, Hayase Y, Kitagawa H. Mercem: Method Name Recommendation Based on Call Graph Embedding[C]. *2019 26th Asia-Pacific Software Engineering Conference*, 2019: 134-141.

[76] Wheeldon R, Counsell S. Power Law Distributions in Class Relationships[C]. *The Third IEEE International Workshop on Source Code Analysis and Manipulation*, 2003: 45-54.

[77] Agrawal A, Menzies T. Is “Better Data” Better Than “Better Data Miners”?[C] *2018 IEEE/ACM 40th International Conference on Software Engineering*. 2018: 1050-1061.

[78] He H B, Garcia E A. Learning from Imbalanced Data[J]. *IEEE Transactions on Knowledge and Data Engineering*, 2009, 21(9): 1263-1284.

[79] Zhang X, Li Y. An empirical study of learning from imbalanced data[C]. *The Twenty-Second Australasian Database Conference-Volume* 115. 2011: 85-94.

[80] López V, Fernández A, García S, et al. An Insight into Classification with Imbalanced Data: Empirical Results and Current Trends on Using Data Intrinsic Characteristics[J]. *Information Sciences*, 2013, 250: 113-141.

[81] Chawla N V, Bowyer K W, Hall L O, et al. SMOTE: Synthetic Minority Over-Sampling Technique[J]. *Journal of Artificial Intelligence Research*, 2002, 16: 321-357.

[82] Breiman L. Random Forests[J]. *Machine Learning*, 2001, 45(1): 5-32.

[83] D'Ambros M, Lanza M, Robbes R. An Extensive Comparison of Bug Prediction Approaches[C]. *2010 7th IEEE Working Conference on Mining Software Repositories*, 2010: 31-41.

[84] MacQueen J. Some methods for classification and analysis of multivariate observations[C]. *The fifth Berkeley symposium on mathematical statistics and probability*. 1967, 1(14): 281-297.

[85] Wen Z H, Tzerpos V. An Effectiveness Measure for Software Clustering Algorithms[C]. *The 12th IEEE International Workshop on Program Comprehension*, 2004, 2004: 194-203.

刘靖雯 于 2019 年在西安交通大学自动化专业获得学士学位。现在西安交通大学控制科学与工程专业攻读硕士学位。研究领域为软件缺陷预测。研究兴趣包括: 软件架构。Email: ljw934058229@stu.xjtu.edu.cn

晋武侠 于 2020 年在西安交通大学计算机科学与技术专业获得博士学位。现在西安交通大学软件学院任助理教授。主要研究领域包括: 软件设计结构、软件质量、微服务。Email: jinwuxia@mail.xjtu.edu.cn

屈宇 于2015年获得西安交通大学系统工程专业博士学位, 现为美国加州大学河滨分校计算机科学与工程系博士后。研究领域为软件安全、可信软件。研究兴趣包括: 深度学习、机器学习、复杂网络等。Email: yu.qu@ucr.edu。

金洋旭 于 2019 年在西安交通大学计算机科学与技术专业获硕士学位。现任中国银行软件中心初级软件工程师。主要研究领域包括: 软件结构分析、可信软件。Email: jinyx7261@mail.notes.bank-of-china.com

范铭 于2019年1月在西安交通大学计算机专业获得博士学位, 并于2019年6月在香港理工大学计算机专业获得博士学位(双学位)。现任西安交通大学电信学部网络空间安全学院助理教授。研究领域为软件安全、AI 安全。研究兴趣包括: AI 可解释性分析。Email: mingfan@mail.xjtu.edu.cn