软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software, 2023,34(6):2509−2525 [doi: 10.13328/j.cnki.jos.006846]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

GKCI: 改进的基于图神经网络的关键类识别方法*

周纯英[1], 曾 诚[1,3], 何 鹏[1,2,3], 张 龑[1,3]

[1](湖北大学 计算机与信息工程学院, 湖北 武汉 430062)

[2](湖北大学 网络空间安全学院, 湖北 武汉 430062)

[3](湖北省教育信息化工程技术研究中心, 湖北 武汉 430062)

通信作者: 何鹏, E-mail: penghe@hubu.edu.cn

摘 要: 研究人员将软件系统中的关键类作为理解和维护一个系统的起点, 而关键类上的缺陷给系统带来了极大的安全隐患. 因此, 识别关键类可提高软件的可靠性和稳定性. 常用的识别方法是将软件系统抽象为一个类依赖网络, 再根据定义好的度量指标和计算规则计算每个节点的重要性得分, 如此基于非训练框架得到的关键类, 并没有充分利用软件网络的结构信息. 针对这一问题, 基于图神经网络技术提出了一种有监督的关键类识别方法. 首先, 将软件系统抽象为类粒度的软件网络, 并利用网络嵌入学习方法 Node2Vec 得到类节点的表征向量, 再通过一个全连接层将节点的表征向量转换为具体分值; 然后, 利用改进的图神经网络模型, 综合考虑类节点之间的依赖方向和权重, 进行节点分值的聚合操作; 最后, 模型输出每个类节点的最终得分并进行降序排列, 从而实现关键类的识别. 在 8 个 Java 开源软件系统上, 通过与基准方法的实验对比, 验证了该方法的有效性. 实验结果表明: 在前 10 个候选关键类中, 所提方法比最先进的方法提升了 6.4%的召回率和 3.5%的精确率.

关键词: 关键类识别; 软件网络; 图神经网络; 软件度量

中图法分类号: TP311

中文引用格式: 周纯英, 曾诚, 何鹏, 张龑. GKCI: 改进的基于图神经网络的关键类识别方法. 软件学报, 2023, 34(6): 2509–2525. http://www.jos.org.cn/1000-9825/6846.htm

英文引用格式: Zhou CY, Zeng C, He P, Zhang Y. GKCI: An Improved GNN-based Key Class Identification Method. Ruan Jian Xue Bao/Journal of Software, 2023, 34(6): 2509−2525 (in Chinese). http://www.jos.org.cn/1000-9825/6846.htm

GKCI: An Improved GNN-based Key Class Identification Method

ZHOU Chun-Ying[1], ZENG Cheng[1,3], HE Peng[1,2,3], ZHANG Yan[1,3]

[1](School of Computer Science and Information Engineering, Hubei University, Wuhan 430062, China)

[2](School of Cyber Science and Technology, Hubei University, Wuhan 430062, China)

[3](Engineering Technology Research Center for Education Informatization of Hubei Province, Wuhan 430062, China)

Abstract: Researchers use key classes as starting points for software understanding and maintenance. These key classes may cause a significant security risk to the software if they have defects. Therefore, identifying key classes can improve the reliability and stability of the software. Most of the existing methods are based on non-trainable solutions, which calculate the score of each node according to a certain calculation rule, and cannot fully utilize the structural information available in the software network. To solve these problems, a supervised deep learning method is proposed based on graph neural network technology. First, the project is built as a software network and the network embedding learning method Node2Vec is used to learn the node representation. Then, the node representation is mapped

* 基金项目: 国家自然科学基金(62102136); 湖北省重点研发计划(2021BAA184, 2021BAA188, 2022BAA044); 湖北省技术创新专项(2020AEA008)

本文由“软件可信性与供应链安全前沿进展”专题特约编辑向剑文教授、郑征教授、申文博研究员、常瑞副教授、田聪教授推荐.

收稿时间: 2022-09-05; 修改时间: 2022-10-10; 采用时间: 2022-12-14; jos 在线出版时间: 2023-01-13

into a score through a simple dense network. Second, the aggregation function of the graph neural networks (GNNs) is improved to aggregate important scores instead of node embedding. The direction and weight information between nodes are also considered when aggregating the scores of neighbor nodes. Finally, the nodes are ranked in descending order according to the predicted score output by the model. To evaluate the effectiveness of the proposed method, it is applied to eight Java open-source software systems. The experimental results show that the proposed method performs better than benchmark methods. In the top 10 key candidates, the proposed method achieves 6.4% higher recall and 3.5% higher precision than the state-of-the-art.

Key words: key class identification; software network; graph neural network (GNN); software measurement

随着软件系统在其生命周期中的不断调整和发展, 理解软件系统是开发人员修改和维护好系统的关键一步. 然而, 大多数软件在后期维护工作中会缺乏详细的交互文档, 加大了维护人员对软件理解的难度. LaToza 等人[1]提出: 从软件系统的核心(即关键类)着手, 可以高效地理解软件项目. 关键类与系统其他模块紧密耦合, 关联着系统的大部分功能, 可以在更高的抽象级别上帮助理解系统的设计. 此外, 关键类在软件系统中的结构和功能都比其他模块更重要, 一旦出现缺陷, 根据模块之间的级联和传播效应, 可能会导致与其有耦合关系的模块也受到影响, 甚至导致整个软件系统瘫痪[2]. 因此, 有效识别软件系统中的关键类, 不仅有助于工程师对系统的理解, 也能帮助管理人员有侧重地关注潜在的安全隐患, 降低后期维护成本, 提高软件可靠性和稳定性.

在软件工程领域, 早期主要是根据类的内部复杂性度量属性来识别关键类, 如通过代码行的规模[3]、类中的方法数量[4]、类的继承深度[5]等. 但是, 该类方法忽略了软件系统中类之间的外部结构属性. 在一个软件系统中, 每个类并不是孤立存在的, 而是以各种依赖关系形成一个网络结构, 即软件网络或类依赖网络, 其中, 关键类往往对网络的结构和功能影响较大[6]. 复杂网络理论兴起后, 软件被抽象为一种人工复杂网络[7], 关键类识别则可转换为复杂网络中节点重要性排序问题, 这为软件系统中关键类识别提供了一个新视角.

评估复杂网络中节点重要性的方法有很多, 包括度中心性(degree centrality, DC)[8]、介数中心性(betweenness centrality, BC)[8,9]、H 指数(H-index)[10]、K-shell[11]、PageRank[12,13]等等. DC 是一种测量节点中心性的简单方法, 节点的度越大, 说明该节点越重要, 但却没有考虑网络的全局结构. BC 根据节点通过最短路径与其他节点的连通性来衡量节点的重要性, 但是计算复杂度很高. H 指数广泛用于根据论文引用量来量化期刊或研究人员的影响力和声誉, 扩展到网络中, 用来测量节点对其邻域的影响程度. K-shell 关注节点的位置, 但无法提供量化的方法来衡量节点的重要性. PageRank 使用邻居的信息进行迭代, 以评估节点的影响, 但对随机网络非常敏感. 不难发现: 以上技术基于一个非训练的固定框架, 是根据定义好的度量指标和计算规则来计算每个节点的重要性, 基本不涉及参数的优化学习, 不能灵活地适应各种网络结构.

目前, 图神经网络(graph neural network, GNN)受到了越来越多的关注, 并在节点分类[14−17]和图分类[18,19]任务上取得了显著成效. GNNs 旨在利用图结构和节点或边的属性信息来学习节点或图的表示, 并用于下游任务中. 然而, 现有的 GNNs 大多是基于向量的嵌入聚合, 而学习出来的节点特征向量实际上不能直观地表示节点的重要性. 为了解决这一问题, Park 等人[20]提出一种 GENI 方法, 该方法将 Node2Vec[21]获得的节点特征映射为节点的重要性分值, 并运用图注意力网络(graph attention network, GAT)[22]方法自适应地聚合邻居节点的分值, 在评估节点重要性方面取得了不错的效果.

本文通过改进 GNN 模型, 提出一种新的关键类识别方法(GNN-based key class identify, GKCI). 其过程大致如下: 首先, 利用网络嵌入学习提取类依赖网络中每个类节点的表征向量; 接着, 采用一个网络映射将节点的表征向量转换为一个标量, 作为节点的初始分值; 然后, 对目标节点的邻居分值进行聚合, 考虑不同邻居对目标节点的影响不同, 在聚合时, 将方向和权重也纳入计算; 最后, 根据节点的得分进行降序排序, 从而实现关键类的识别.

本文针对软件系统中关键类识别任务展开研究, 主要贡献可归纳如下.

(1) 改进了 GNN 模型的邻域聚合函数, 即, 通过聚合目标节点邻居的分值来代替以往的向量聚合, 提出了一种可更直接捕获节点及其邻居重要性的有监督的深度学习方法, 以提升软件系统中关键类的识别效果;

(2) 探析了软件系统中类依赖关系的方向和权重信息的影响, 并在 8 个开源软件上进行了实证分析, 发现权重比方向在关键类识别上更具有参考价值, 且本文所提 GKCI 方法在有向加权情境下, 识别效果比基准方法(DC[8]、BC[8,9]、H-index[10]、K-shell[11]、PageRank[12,13]、文献[23]和文献[24])具有更显著的提升.

本文第 1 节概括关键类识别及类似问题在相关领域中的研究现状. 第 2 节为理论基础. 第 3 节、第 4 节分别是本文 GKCI 方法的介绍和实验及结果分析. 第 5 节主要讨论文中实验处理问题、应用价值以及不足之处. 第 6 节是全文工作总结.

1 相关工作

软件网络是根据复杂网络理论把软件系统抽象为一个网络, 从网络视角研究软件系统, 有助于优化软件设计结构、提高软件质量. 在复杂网络中, 称对网络影响较大的节点为关键节点. 软件系统中同样存在对整体影响较大的少数模块, 根据它们的依赖调用关系和耦合紧密程度, 可轻易地给整个软件结构带来重大的影响. 在软件测试方面也发现, 由于模块间的级联效应, 往往在部分模块出现故障时, 甚至导致整个软件系统发生故障. 因此, 有效且准确地识别软件系统中这些关键模块, 对提升软件系统的质量具有重大意义.

现有大多数关键类识别方法都基于复杂网络理论来度量节点的重要性, 属于静态分析. 潘伟丰等人[13,25]先后在包粒度和类粒度的软件网络上, 提出了基于加权 PageRank 算法的关键包/类识别方法. Pan 等人[26]提出了一种广义 k 核分解方法, 用于计算每个类的广义核度, 根据广义核数对类进行排序, 并将排名靠前的视为候选关键类. Şora 等人[27]从代码中提取多种类属性, 并研究对比了这些属性对关键类识别的适合性. Li 等人[28]提出了一阶结构熵(OSE)度量, 并用于计算节点的重要性. Du 等人[23]提出了一种基于偏好聚合的关键类识别方法(COSPA), 该方法使用 Kemeny-Young[29−31]方法找到一个偏好, 该偏好将 3 种关键类识别方法(即 a-index、InDeg 和 OSE)返回的不同序列对之间的总差异最小化, 并根据这种偏好集计算节点的重要性.

近些年来, 随着深度学习的不断发展, 图数据得以使用深度学习范式进行学习, 为网络中节点重要性排序提供了又一种新思路. Fan 等人[32]设计了一个基于编码器-解码器的图神经网络框架, 利用编码器将网络结构中的每个节点表示为一个嵌入向量, 再通过解码器将向量转换为标量, 并以节点的介数中心性 BC 值作为重要性标准进行学习训练, 弥补了 BC 指标在大型网络上计算耗时的不足, 但该方法准确率的上限是 BC. 在该工作基础上, 张健雄等人[24]进一步提出一种改进方法, 采用节点收缩法[33]衡量节点的排序结果. 虽然该方法[24]在软件系统关键类识别上取得了不错的结果, 但是张健雄等人并没有考虑软件网络中边的方向, 并且邻域聚合过程参考了文献[32]的处理方式, 即, 对节点的嵌入向量进行聚合. Park 等人[20]提出一种解决知识图谱领域中节点重要性评估问题的方法 GENI, 该方法基于 GAT 而加以改进, 将节点的初始信息转换为重要性分数, 并使用灵活的中心性调整来聚合重要性分数, 而不是聚合节点嵌入. 得益于监督学习框架和图神经网络, GENI 实现了比无监督算法更高的性能.

对于软件关键类识别研究, 主要难点有: (1) 目前大多数关键类识别方法都是基于非训练的框架, 鲜有研究使用深度学习框架自动学习软件网络的信息; (2) 软件网络建模过程中, 往往只单一地考虑边的方向或权重信息; (3) 现有的图神经网络模型在邻域聚合过程中都是对节点的嵌入向量进行聚合, 难以直接反映邻居节点的差异程度对目标节点的影响.

针对上述问题, 本文提出一种改进的基于 GNN 的关键类识别方法 GKCI, 采用节点分值代替嵌入向量的邻域聚合, 且通过节点中心性灵活地优化聚合分值. 此外, 结合方向与权重, 本文考虑了 4 种情境下的软件网络模型, 即无向无权网络、无向加权网络、有向无权网络和有向加权网络, 并重点围绕以下两个问题进行探讨.

RQ1: GKCI 在哪种情境下的软件网络上, 关键类识别效果更好?

软件建模过程中, 类之间交互的方向和权重信息, 例如, 类间调用的方向性以及调用次数等, 是否有助于提高对软件系统的表征学习?

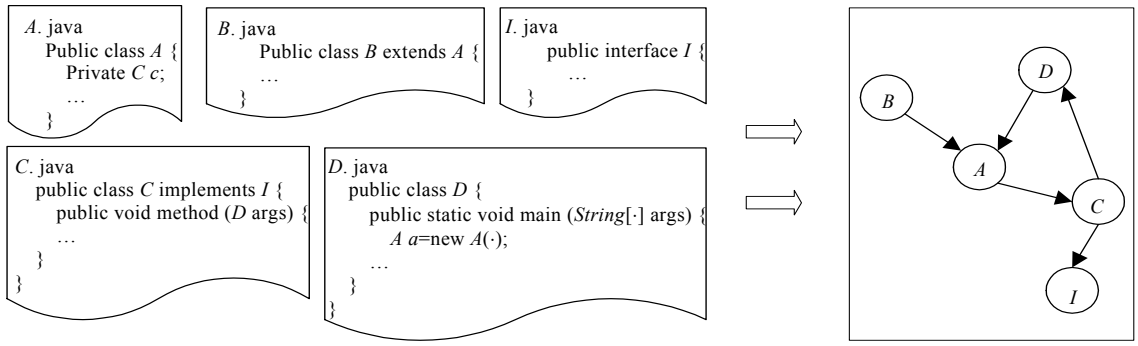
RQ2: GKCI 是否优于已有的关键类识别方法?

图神经网络模型在复杂网络研究领域被证明有效, 相比之下, 在软件工程领域, 该模型对软件系统中关键类预测是否仍然有效?

2 理论基础

2.1 软件网络建模

软件网络是根据软件系统中元素(包括包、类、特征等)之间的依赖关系构建的一种网络, 也称为依赖网络(dependency network, DN). 借助 DependencyFinder API (https://sourceforge.net/projects/depfind/)解析编译后的软件源代码文件, 可以得到元素之间的各种依赖关系. 图 1(a)是 5 个 Java 文件代码片段, 图 1(b)是它们之间对应的类依赖网络, 网络中的每个节点代表一个类, 边为两个类节点之间的依赖关系. 例如, 类 B 是类 A 的子类, 根据两者的继承关系, 则存在一条由 B 指向 A 的连边($B\to A$), $C\to I$ 代表接口实现关系, $C\to D$ 代表参数类型依赖关系, $A\to C$ 和 $D\to A$ 代表聚合关系.



(a) Java 代码片段 (b) 类依赖网络

图 1 一个简单的类依赖网络 CDN (class dependency network)

2.2 图神经网络

GNN 模型[34]对图数据具有强大的表征能力, 可同时利用图结构和节点属性信息来学习节点的表征向量, 生成的节点表征向量可用于各种下游任务. 对于给定的网络 $G=(V,E)$, 分为拓扑结构信息和节点属性信息两部分, 分别用邻接矩阵 A 和节点特征矩阵 X 表示. 邻接矩阵 A 是全局共享的, 不会发生变化; 而特征矩阵 X 在每层都会更新. GNNs 通过聚合节点的邻居特征来迭代更新其自身的特征(如图 2 所示), 节点的表征向量更新为

$$\boldsymbol{h}_v^{(k)}=\sigma(\boldsymbol{W}^k\cdot CONCAT(\boldsymbol{h}_v^{(k-1)},AGGREGATE(\{\boldsymbol{h}_u^{(k-1)}:u\in N(v)\}))) \tag{1}$$

其中, $\boldsymbol{h}_v^{(k)}$ 表示节点 v 在第 k 层的特征向量, $N(v)$表示节点 v 的邻居集合, $\boldsymbol{W}^k$ 是第 k 层的权重矩阵, σ是非线性激活函数, $CONCAT(\cdot)$是拼接函数, $AGGREGATE(\cdot)$是聚合函数.

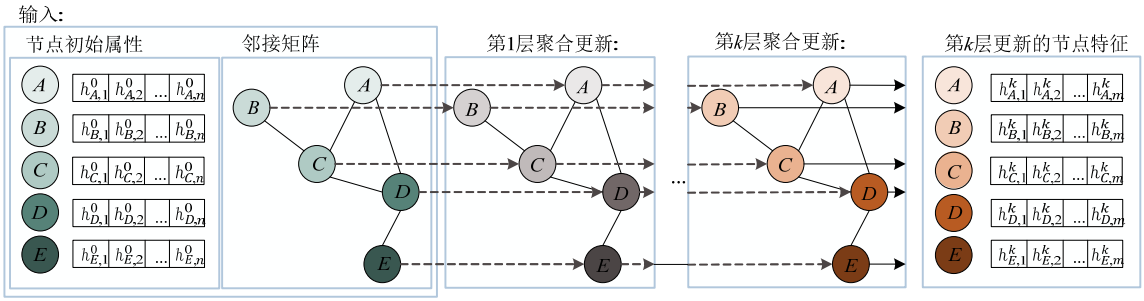


图 2 GNN 的更新迭代架构(n 是节点初始属性的维度, m 是第 k 层节点特征维度)

本文对常规的聚合方式进行改进, 以实现直接建模节点与其邻居节点之间的重要性关系. 此外, 借鉴文献[35]中提出的 GraphSAGE 方法的思想, 本文为每个节点采样固定大小的邻居节点, 以降低算法的空间和时间复杂性. 具体细节见第 3.2 节.

3 研究方法

图 3 给出了本文的整体框架, 包含 3 个阶段: 数据处理、模型构建和关键类识别. (1) 数据处理阶段包括类依赖网络建模和网络嵌入学习; (2) 模型构建阶段主要完成 GKCI 模型的训练, 包括将学习到的节点表征向量映射为具体得分, 并通过聚合层更新分值, 以及对节点得分进行预测; (3) 关键类识别阶段主要负责将模型输出的节点得分进行排序, 评价关键类的识别准确情况.

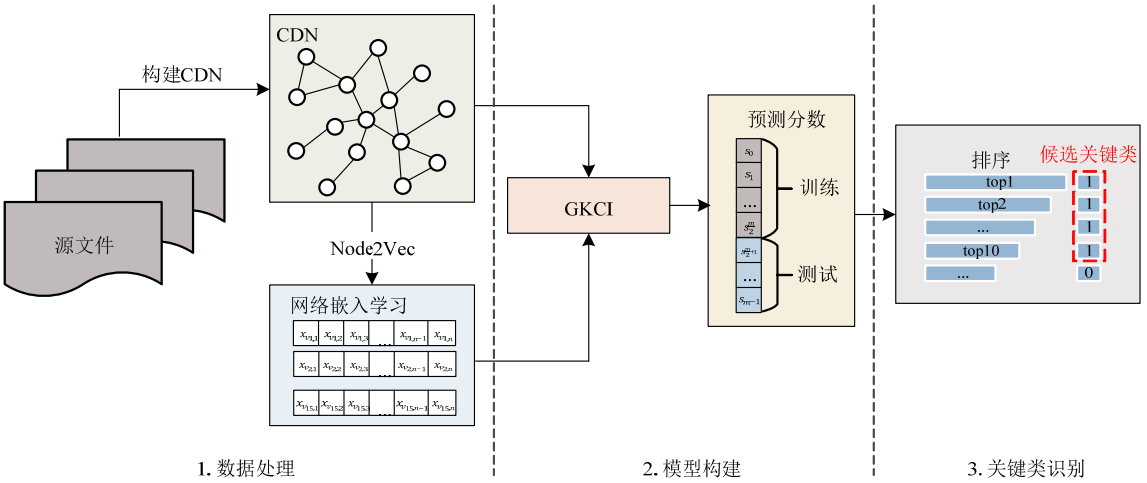


图 3 本研究的整体框架

3.1 数据处理

3.1.1 类依赖网络建模

本文旨在对软件系统中的关键类进行识别, 所以仅构建类粒度级别的软件网络, 即以类(class)为节点、类之间关系为连边的类依赖网络, 具体定义如下:

$$CDN=(V,E,W) \tag{2}$$

其中, V 表示网络中节点的集合, 节点 v_i ($v_i \in V$)表示软件系统中的一个类或接口; E 表示网络中边的集合, 连边 e_{ij} ($e_{ij}=\langle v_i,v_j\rangle \in E$)表示节点 v_i 与节点 v_j 之间存在依赖关系; W 表示连边权重的集合, 权值 w_{ij} ($w_{ij}=\langle v_i,v_j\rangle \in W$)表示 e_{ij} 边上的权重. 正如文献[36,37]中的处理方式, 在 CDN 构建过程中, 本文类节点的依赖关系主要考虑以下 3 种情况.

(1) 继承: 假如类 c_1 继承于类 c_2, 或实现接口 c_2, 则存在有向边 $e_{12}=\langle v_{c_1},v_{c_2}\rangle$;

(2) 聚合: 假如类 c_1 包含类 c_2 的属性, 则存在有向边 $e_{12}=\langle v_{c_1},v_{c_2}\rangle$;

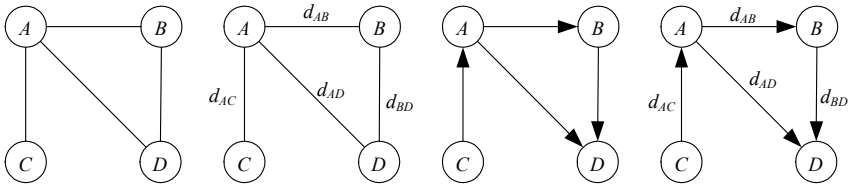
(3) 参数: 假如类 c_1 的方法调用了类 c_2 的方法, 则存在有向边 $e_{12}=\langle v_{c_1},v_{c_2}\rangle$.

如图 4 所示, 本文考虑 4 种情境, 分别是无向无权、无向加权、有向无权和有向加权. 其中, 类之间的相互依赖次数代表了类之间的影响程度. 本文对连边权重的计算方式表示如下:

$$w_{ij}=\frac{d_{ij}}{\sum_{k\in N(j)} d_{jk}} \tag{3}$$

其中, w_{ij} 表示节点 v_i 和节点 v_j 之间的权重; d_{ij} 表示两节点之间的依赖次数, 即若类之间存在以上 3 种依赖关系之一(聚合、继承、调用), 则 d_{ij} 增加 1; $N(j)$表示节点 v_j 的邻居集合. 对于无权网络, $d_{ij}=1$, $w_{ij}=1$.

值得一提的是: 在加权网络中, 边的权重与方向无关, 而与当前的目标节点有关. 例如在图 4(b)中: 当以节点 A 为中心聚合其邻居节点时, 由于节点 B 也有自己的邻居, 且对每个邻居节点的贡献值并不相同, 则 w_{AB} 仅代表节点 B 对节点 A 的贡献值, 即 $w_{AB}=\dfrac{d_{AB}}{\sum_{k\in N(B)}d_{Bk}}=\dfrac{d_{AB}}{d_{AB}+d_{BD}}$; 同理, 当以节点 B 为中心聚合其邻居节点时, w_{BA} 代表节点 A 对节点 B 的贡献值, 即 $w_{BA}=\dfrac{d_{AB}}{\sum_{k\in N(A)}d_{Ak}}=\dfrac{d_{AB}}{d_{AB}+d_{AC}+d_{AD}}$. 这反映了节点之间的相互影响存在一定的差异, 并不完全等价.



(a) 无向无权图 (b) 无向加权图 (c) 有向无权图 (d) 有向加权图

图 4 4 种情境下的类依赖网络

3.1.2 网络嵌入学习

网络嵌入旨在将网络中的节点映射为低维向量表示, 并保留网络的拓扑结构信息. 本文采取应用广泛的 Node2Vec[21]作为节点嵌入学习方法. 该方法是一种综合了广度优先遍历(BFS)和深度优先遍历(DFS)的嵌入方法, 通过引入参数 p 和 q, 灵活控制 BFS 和 DFS 的决策. 参数 p 是返回概率, 控制重复访问刚刚访问过的节点概率. 参数 q 是出入参数, 如图 5 所示: 假设当前随机游走经过边(t,v)到达节点 v, 如果 q>1, 则游走过程中会倾向于在起始节点 t 周围采样, 即 BFS 策略; 如果 q<1, 则会倾向于远离节点 t 处采样, 即 DFS 策略; 当 p=1, q=1 时, 则等价于随机游走. 给定当前节点 v, 其访问的上一节点是 t, 则访问下一个节点 x 的概率为

$$\alpha_{pq}(t,x)=\begin{cases}\dfrac{1}{p}, & \text{if } d_{tx}=0\\ 1, & \text{if } d_{tx}=1\\ \dfrac{1}{q}, & \text{if } d_{tx}=2\end{cases} \tag{4}$$

其中, d_{tx} 表示节点 t 与 x 之间的最短路径. 图 5 中: 节点 x_1 与节点 t 之间的最短路径为 1, 则访问节点 x_1 的概率α=1; 节点 x_2 和 x_3 与节点 t 之间的最短路径为 2, 则访问节点 x_2 和 x_3 的概率 $\alpha=\dfrac{1}{q}$, 重复访问上一节点 t 的概率 $\alpha=\dfrac{1}{p}$.

网络嵌入的目标是: 通过函数 f, 将 $CDN=(V,E,W)$中的节点映射为一个低维向量, 即 $f:v_i\to x_i\in R^d$, $v_i\in V$, $d<<|V|$, 且映射函数 f 保留了节点的邻接结构.

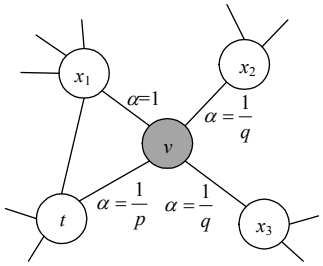


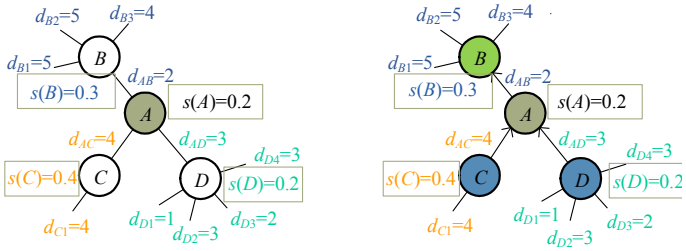
图 5 Node2Vec 中的随机游走过程

3.2 GKCI方法

图 6 展示了 GKCI 方法在加权网络中无向图聚合和有向图聚合的示例. 假设节点 A 需要聚合其邻居节点 B, C 和 D 的分数, $s(A)$、$s(B)$、$s(C)$和 $s(D)$分别代表 4 个节点当前的重要性分数, d_{AB}、d_{AC}和 d_{AD}分别代表节点 A 与其邻居节点 B、C 和 D 之间的依赖次数. d_{B1}、d_{B2}和 d_{B3}分别是节点 B 的其他邻居, 节点 C 和节点 D 同理. 由第 3.1.1 节可知, 依赖次数并不代表边的权重. 根据公式(3), 可分别计算出 3 条边的权重:

$$w_{AB}=\frac{d_{AB}}{d_{AB}+d_{B1}+d_{B2}+d_{B3}}=0.125,\ w_{AC}=\frac{d_{AC}}{d_{AC}+d_{C1}}=0.5,\ w_{AD}=\frac{d_{AD}}{d_{AD}+d_{D1}+d_{D2}+d_{D3}+d_{D4}}=0.25.$$

对于图 6(a), 由于不区分方向, 则直接聚合其邻居分数 $s_N(A)=w_{AB}\cdot s(B)+w_{AC}\cdot s(C)+w_{AD}\cdot s(D)=0.2875$, 然后将得到的 $s_N(A)$与节点 A 自身的分数 $s(A)$作拼接, 并经过激活函数后得到节点 A 更新后的分数. 对于图 6(b), 根据边的方向, 将节点 A 的邻居划分为前继邻居节点和后继邻居节点, 需要分别聚合其前继邻居节点和后继邻居节点的分数: 若节点 C 和 D 为节点 A 的前继邻居节点, 则 $s_{N_{in}}(A)=w_{AC}\cdot s(C)+w_{AD}\cdot s(D)=0.25$; 若节点 B 为后继邻居节点, 则 $s_{N_{out}}(A)=w_{AB}\cdot s(B)=0.0375$. 然后, 将得到的 $s_{N_{in}}(A)$ 和 $s_{N_{out}}(A)$ 与节点 A 自身的分数 $s(A)$作拼接, 并经过激活函数后, 得到节点 A 更新后的分数. 同理, 无权网络的聚合方式也是如此, 只是边依赖次数和权重都为 1.



(a) 无向加权图聚合 (b) 有向加权图聚合

图 6 GKCI 聚合示例

根据上述示例, 可以更清楚地了解 GKCI 的聚合过程. 本文旨在构建一个简单的图神经网络架构, 主要包括 1 个节点得分映射层、1 个邻域节点分数聚合层和 1 个中心性调整机制. 如图 7 所示: 对于节点 v, 根据数据处理阶段生成的初始嵌入向量 $\boldsymbol{x}_v\in R^d$, 其中, d 是网络嵌入学习输出的向量维度, 则节点 v 的初始分值为

$$s^0(v)=ScoringNet(\boldsymbol{x}_v) \tag{5}$$

其中, *ScoringNet* 参照了文献[20]中的设置, 使用一个简单的全连接神经网络, 根据输入的嵌入向量返回一个标量分值. 注意, *ScoringNet* 与模型的其他部分将共同参与训练.

对节点 v 的邻居进行固定数量的随机采样, 并聚合采样后的邻居分值. 对于无向图, 聚合方式如下(如图 7(a)所示):

$$s_N^l(v)=\sum_{u\in N(v)}w_{vu}s^{l-1}(u) \tag{6}$$

$$s^l(v)=\sigma(CONCAT(s_N^l(v),s^{l-1}(v))) \tag{7}$$

其中, $N(v)$是节点 v 采样后的邻居节点集合, w_{vu} 是连边 e_{vu} 的权重, $s^l(v)$是节点 v 在第 l 层的分值, $CONCAT(\cdot)$是拼接函数, σ是激活函数.

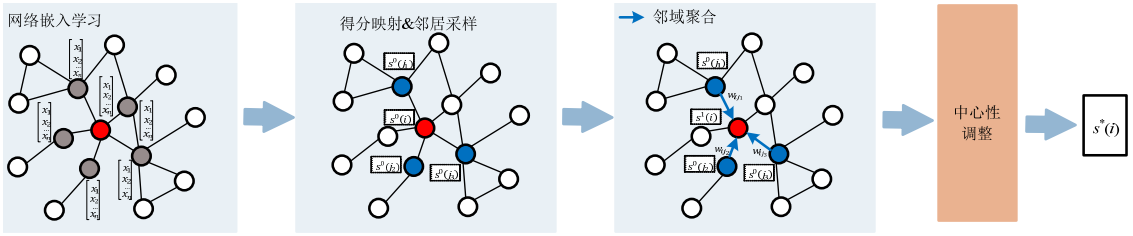
对于有向图, 在聚合邻居节点时, 需要将前继邻居节点和后继邻居节点的分数分别进行聚合(如图 7(b)所示):

$$s_{N_{in}}^l(v)=\sum_{u\in N_{in}(v)}w_{uv}s^{l-1}(u) \tag{8}$$

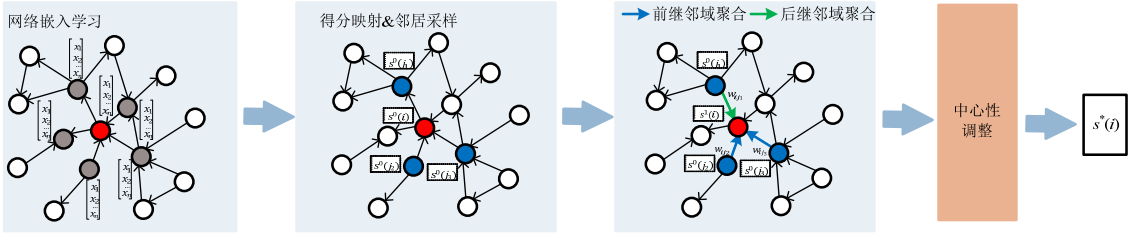
$$s_{N_{out}}^l(v)=\sum_{u\in N_{out}(v)}w_{vu}s^{l-1}(u) \tag{9}$$

$$s^l(v)=\sigma(CONCAT(s^l_{N_{in}}(v),s^l_{N_{out}}(v),s^{l-1}(v))) \tag{10}$$

其中，$N_{in}(v)$是节点 v 采样后的前继邻居节点集合，$N_{out}(v)$是节点 v 采样后的后继邻居节点集合，$s^l_{N_{in}}(v)$ 和 $s^l_{N_{out}}(v)$ 分别是节点 v 聚合其前继邻居和后继邻居分值后的结果.



(a) 无向图聚合



(b) 有向图聚合

图 7　GKCI 模型架构

正如文献[20]所言：通常，节点的重要性与其在图中的中心性正相关. 因此，本文也引入这种先验知识来对模型进行调整:

$$c(v)=\log(d(v)+\varepsilon) \tag{11}$$

其中，$c(v)$是节点 v 的中心性，$d(v)$是节点 v 的度或者入度，ε是一个正常数.

为了减少节点中心性与实际输入的重要性得分之间可能存在的偏差，使用缩放和移位的方法作为节点中心性的调整:

$$c^*(v)=\gamma c(v)+\beta \tag{12}$$

其中，γ和β是用于缩放和移位的可学习参数.

我们将中心性调整应用在最后一层的输出上，因此，最终模型输出的节点得分为

$$s^*(v)=\sigma(c^*(v)\cdot s^l(v)) \tag{13}$$

其中，$s^l(v)$是模型最后一层的分数，σ是一个非线性激活函数. 算法 1 给出了 GKCI 的整体描述.

算法 1. GKCI 算法.

输入：类依赖网络 $CDN=(V,E,W)$，节点初始特征向量$\{x_v,\forall v\in V\}$，邻居节点集合$\{N(v),\forall v\in V\}$；

输出：每个节点的预测分值 s_v.

1. 将每个节点 v 的初始特征向量采用公式(5)进行转换，得到初始分数 s_v^0;
2. 如果 CDN 是无向图
3. 　对每个节点 v 进行公式(6)、公式(7)的邻居分数聚合处理;
4. 否则
5. 　对每个节点 v 进行公式(8)–公式(10)的邻居分数聚合处理;
6. 对每个节点 v 进行公式(11) –公式(13)的中心性调整，得到节点的最终预测分数 s_v;
7. 输出 s_v.

3.3 关键类识别

为了准确预测节点的重要性分值, 本文采用二分类交叉熵损失函数(binary CrossEntropy loss)来训练模型:

$$Loss = \frac{1}{|V|}\sum_{v\in V} -\omega(y_v \cdot \log(s^*(v)) + (1-y_v)\cdot \log(1-s^*(v))) \tag{14}$$

其中, $s^*(v)$是节点 v 的对应得分; y_v是节点 v 的真实标签, 0 代表非关键类, 1 代表关键类; ω是超参数权重值, 一般设置为 1. 为了更好地评价模型的预测结果, 我们分别将测试集中预测值排名 top 10、top 15 和 top 20 的类视为候选关键类进行分析, 即候选关键类的标签记为 1, 非候选类的标签记为 0.

4 实验与结果分析

4.1 数据集

为了更好地开展验证分析, 参考表 1 所示实验数据集, 本文选取 8 个本领域公开常用的开源软件系统作为实验对象, 其中, 最后两列为对应类依赖网络的节点数量和实际关键类数. 表 2 展示了每个项目的真实关键类集.

表 1 实验数据集

系统	版本	URLs	节点总数	关键类总数
Ant	1.6.1	http://ant.apache.org/	482	10
Hibernate	5.2.12	http://hibernate.org/orm/releases/5.2/	3 754	14
jEdit	5.1.0	https://sourceforge.net/projects/jedit/files/jedit/5.1.0/	992	7
JGAP	3.6.3	http://sourceforge.net/projects/jgap/	423	18
JHotDraw	6.0b.1	http://sourceforge.net/projects/jhotdraw/	516	9
JMeter	2.0.1	https://archive.apache.org/dist/jakarta/jmeter/	509	14
Log4j	2.10.0	https://archive.apache.org/dist/logging/log4j/2.10.0/	848	9
Wro4J	1.6.3	http://code.google.com/p/wro4j/	236	12

表 2 已知关键类集(ground truths[25])

系统	关键类(按字母顺序排列)
Ant	ElementHandler IntrospectionHelper Main Project ProjectHelper RuntimeConfigurable Target Task TaskContainer UnknownElement
Hibernate	Column Configuration ConnectionProvider Criteria Criterion Projection Query Session SessionFactory SessionFactoryImplementor SessionImplementor Table Transaction Type
jEdit	Buffer EBMessage EditPane jEdit JEditTextArea Log View
JGAP	AveragingCrossoverOperator BaseGene BaseGeneticOperator BestChromosomesSelector BooleanGene Chromosome Configuration CrossoverOperator DoubleGene FitnessFunction FixedBinaryGene GaussianMutationOperator Genotype IntegerGene MutationOperator NaturalSelector Population WeightedRouletteSelector
JHotDraw	CompositeFigure Drawing DrawingEditor DrawingView DrawApplication Figure Handle StandardDrawingView Tool
JMeter	AbstractAction JMeterEngine JMeterGUIComponent JMeterThread JMeterTreeModel PreCompiler Sampler SampleResult TestCompiler TestElement TestListener TestPlan TestPlanGui ThreadGroup
Log4j	Appender Configuration Filter Layout Logger LoggerConfig LoggerContext StrLookup StrSubstitutor
Wro4J	Group Resource ResourcePostProcessor ResourcePreProcessor ResourceType UriLocator UriLocatorFactory WroFilter WroManager WroManagerFactory WroModel WroModelFactory

4.2 评价指标及基准模型

本文采用常用的召回率(recall)和准确率(precision)两个评价指标, 计算公式如下:

$$precision@K = \frac{TP}{K} \tag{15}$$

$$recall@K = \frac{TP}{KC} \tag{16}$$

其中, TO (true positive)表示 top K 个候选实例中正样本个数, 即预测类别为真实关键类的实例个数; K 表示候选关键类总数; KC (key class)表示实际关键类总数. 换句话说, 召回率为所有真实关键类中被检测出来的比例, 即 top K 中关键类数量除以实际关键类总数, 与候选关键类总数无关; 准确率为所有候选关键类中为真实关键类的比例, 即检索到的关键类数量除以候选关键类总数, 候选关键类总数越大, 准确率越低.

为验证本文所提方法 GKCI 的效果, 我们选取了 5 种基于复杂网络理论的经典计算方法 DC、BC、K-shell、H-index 和 PageRank 作为对照方法. 此外, 本文还选取了 Du 等人[23]提出的 COSPA 方法和张健雄等人[24]提出的基于图自编码器 GAE 的识别方法作为对照. 此外, 本文还使用 Wilcoxon 符号秩检验(p 值)和 Cliff's 影响大小(d 值)[38]来验证不同方法在预测性能上是否具有统计学意义的显著差异, 其中, p 值小于 0.05 表示两组数据具有统计学差异; d 值与影响大小差异的对应关系见表 3.

本文中, d 值为正表示左侧方法的影响大于右侧的方法.

表 3 不同 d 值对应的影响大小程度

影响大小	$\|d\|$
非常小	0.008
小	0.147
适中	0.33
大	0.474
非常大	0.622
巨大	0.811

4.3 实验设置

本文方法的实现基于 Python 以及 PyTorch 框架, 所有实验都在装有 NVIDIA GeForce GTX5000-16G 的服务器上运行. 训练过程中的参数设置见表 4. 本文将根据每个项目的样本比例按 1:1 划分为训练集和测试集, 其中, 关键类的数量在训练集和测试集中各占一半. 为了避免随机划分带来的偏差, 我们随机划分了 10 次, 并对 10 次预测结果取平均值. 本文在聚合层的邻居采样个数设为 10, 即节点的邻域集小于 10, 则采取有放回的采样 10 次, 否则随机采样 10 个邻居节点. 由表 1 可知: 数据集中关键类的平均数量在 10–20 之间, 则测试集中的关键类只有 5 个左右. 因此, 为了便于评价模型的预测结果, 我们分别取预测值排名前 10、前 15 和前 20 的类视为候选关键类.

表 4 模型参数设置

参数名称	参数值
Node2Vec 参数 p	0.25
Node2Vec 参数 q	2
Node2Vec 输出的节点嵌入向量维度 d	32
GKCI 学习率	0.000 1
训练周期(epoch)	1 000
邻居采样个数	10
聚合层数	1
优化器	Adam
dropout	0.1

4.4 结果与分析

RQ1: GKCI 在哪种情境下的软件网络上关键类识别效果更好?

该问题本质上是分析类依赖网络中边的方向和权重信息对关键类识别的影响. 图 8 给出了 8 个实验对象在 4 种网络建模情境下的召回率和准确率的整体结果(箱线图展示一组数据的最大值、上四分位数、中位数、下四分位数和最小值). 具体来说:

- 在@10 的候选关键类中, 整体上, 无向加权和有向加权情境下的平均召回率分别为 0.695 和 0.693, 比无向无权和有向无权情境下的平均召回率分别提高了 0.87% (0.689)和 9.48% (0.633); 而对于准确率, 趋势极为相似, 无向加权和有向加权情境下的平均准确率分别为 0.374 和 0.376, 分别比无向无权和有向无权情境下的平均准确率提高了 1.36% (0.369)和 10.91% (0.339), 且在有向加权情境下的准确率最高;
- 在@15 和@20 的候选关键类中, 整体上, 无向加权和有向加权情境下的平均召回率分别比无权情境下提高了 2.59% (0.734)、11.13% (0.683)和 3.02% (0.761)、13.26% (0.694), 而对应的平均准确率分别提高了 3.80% (0.263)、11.29% (0.248)和 3.36% (0.207)、13.09% (0.191).

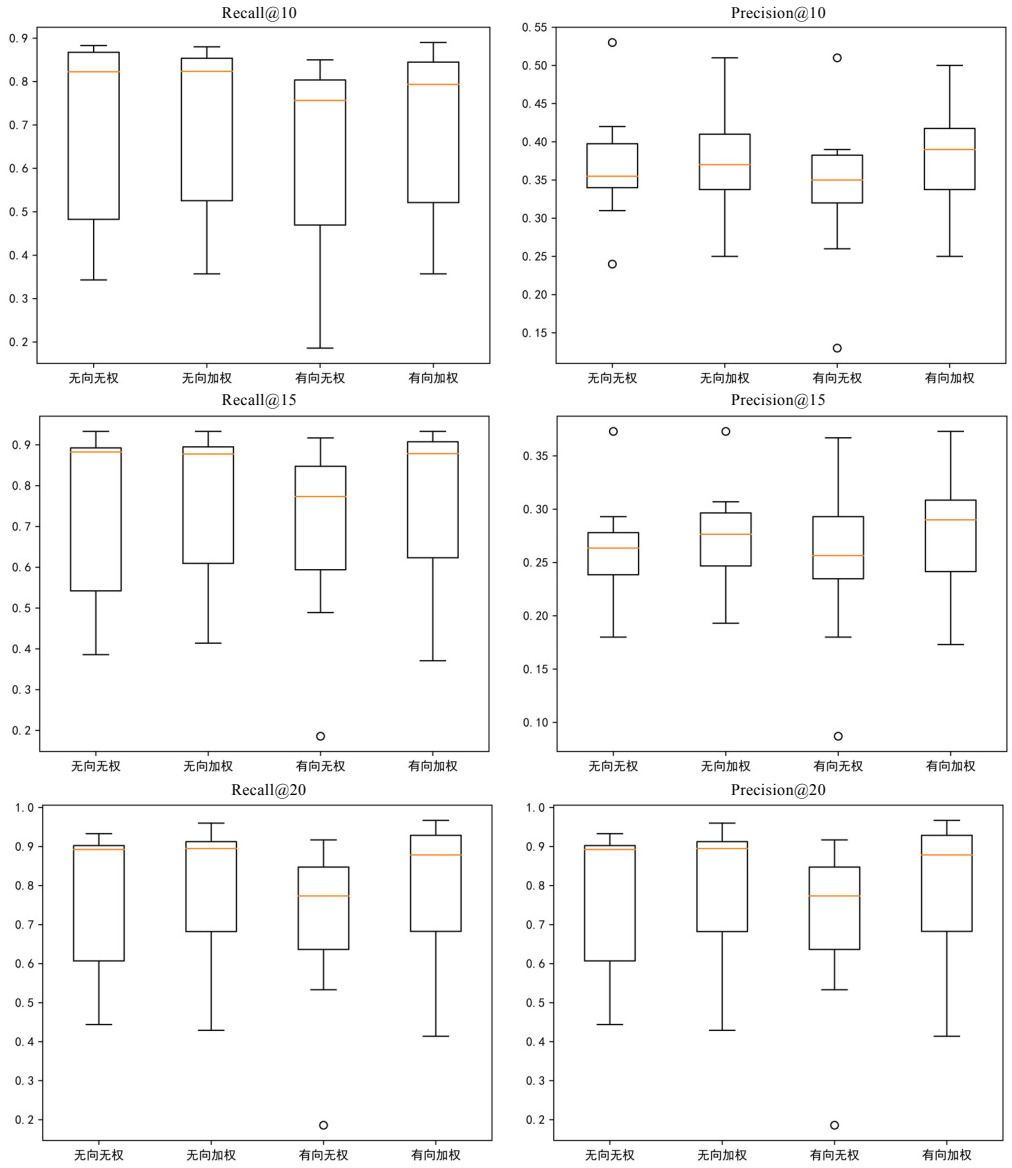


图 8　4 种网络情境下的召回率/精确率对比箱线图

然而, 如图 9 所示(折线图中的每个节点代表在 8 个项目上的平均结果), 不难发现: 随着返回的候选关键类越多, 召回率就越高, 但准确率却在降低. 整体上, GKCI 在有向加权情境下的关键类识别效果更好, 表现为平均召回率和准确率最高, 其次为无向加权情境, 有向无权情境下效果最差.

此外, 表 5 也给出了各种软件网络建模情境下 GKCI 识别效果差异的显著性对比(其中, 粗体数值标记为差异显著, 括号内数值为 d 值, 正数表示左侧网络的影响大于右侧网络). 由 p 值和 d 值的结果可以看出: 无论是召回率还是准确率, 无权与加权之间的差异非常显著, 且表现为与候选关键类选取的规模无关; 相比之下, 无向和有向在统计意义上整体表现为差异不明显. 具体来说, 有向无权与有向加权之间的差异非常显著, 且 d 值为负数表明加权的效果更好; 而其他 3 种组合的对比, 在统计意义上整体表现为差异不明显, 尤其是无向无权与无向加权、无向加权与有向加权两组对比. 总而言之, 在当前问题上, 考虑类之间依赖关系的权重比方向价值更大, 且在有向的同时考虑权重效果最佳, 即 GKCI 在有向加权的软件网络上关键类识别效果最好.

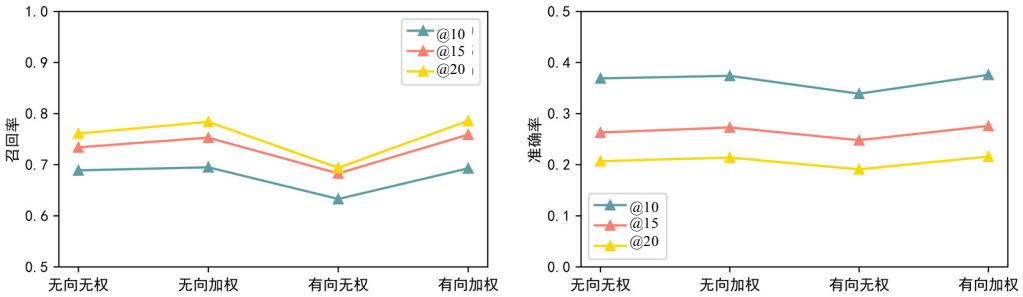


图 9　4 种网络设置的召回率/精确率对比折线图

表 5　4 种网络情境下 Wilcoxon 符号秩检验和 Cliff's 影响大小(Sig.p<0.05, d)

对比	Recall@10	Precision@10	Recall@15	Precision@15	Recall@20	Precision@20
无向 vs.有向	0.059 (0.16)	0.137 (0.035)	0.229 (0.094)	0.438 (0)	0.143 (0.117)	0.307 (0.059)
无权 vs.加权	**0.024*** (–0.117)	**0.030*** (–0.184)	**0.004*** (–0.191)	**0.005*** (–0.219)	**0.004*** (–0.219)	**0.009*** (–0.148)
无向无权 vs.无向加权	0.633 (0.016)	0.47 (–0.047)	0.131 (–0.047)	0.111 (–0.188)	0.179 (–0.094)	0.184 (–0.063)
有向无权 vs.有向加权	**0.019*** (–0.188)	**0.038*** (–0.297)	**0.012*** (–0.344)	**0.024*** (–0.25)	**0.010*** (–0.375)	**0.025*** (–0.219)
无向无权 vs.有向无权	**0.031*** (0.313)	**0.050*** (0.172)	0.161 (0.25)	0.339 (0.063)	0.128 (0.25)	0.27 (0.156)
无向加权 vs.有向加权	0.911 (0.031)	0.802 (–0.078)	0.619 (–0.063)	0.654 (–0.094)	0.835 (0)	0.699 (–0.016)

RQ2: GKCI 是否优于已有的关键类识别方法?

根据 RQ1 结果可知, GKCI 在有向加权情境下效果最佳. 因此, 本小节将重点对比这种情况设置下 GKCI 方法与基准模型在 8 个软件系统上的对比(如表 6、表 7 和图 10 所示).

从实验结果不难看出: 相比 5 种基于复杂网络理论的经典方法, GKCI 占有绝对优势. 具体来说: 在@10 中, GKCI 在召回率上分别比 DC、BC、K-shell、H-index 和 PageRank 平均提高了 19%、28.8%、30.7%、39%和 30.2%, 在准确率上分别平均提高了 12%、17.2%、15.8%、20.3%和 17.2%; 在@15 中, GKCI 在召回率上分别比 DC、BC、K-shell、H-index 和 PageRank 平均提高了 14.7%、27.7%、23.3%、40.7%和 27.3%, 在精确率上分别平均提高了 5.9%、10.9%、7.4%、14.5%和 10.6%; 在@20 中, GKCI 在召回率上分别比 DC、BC、K-shell、H-index 和 PageRank 平均提高了 8.5%、16.9%、19.3%、39.3%和 21.4%, 在精确率上分别平均提高了 2.5%、5.2%、4.2%、10.5%和 6.4%. 此外, 除 K-shell (recall@15, recall@20, precision@20)外, 表 6 和表 7 中的 p 值均小于 0.05 且 d 值整体上都大于 0.474, 进一步说明本文 GKCI 方法相比上述 5 种方法的改进效果更明显.

另外, 我们也与两种最先进的同行方法 COSPA 和 GAE 方法进行了对比. 实验结果显示, GKCI 整体性能都优于 COSPA: 在@10 中, 召回率和准确率分别平均提高了 20.3%和 11%; 在@15 中, 召回率和准确率分别平均提高了 21.4%和 7.4%; 在@20 中, 召回率和准确率分别平均提高了 21.1%和 5.5%. GCKI 在 Ant 和 jMeter 上的性能也优于 GAE: 在@10 中, 召回率和准确率分别平均提高了 6.4%和 3.5%; 在@15 中, 召回率和准确率分别平均提高了 6.5%和 5.6%; 在@20 中, 召回率和准确率分别平均提高了 7.4%和 2%. 同样, 由表 6 和表 7 中的 p 值可以看出, GKCI 相比 COSPA 在召回率和准确率上的改进也具有统计意义上的显著性; 而相比 GAE 并不显著, 但 d=0.5 的结果显示, GKCI 的影响大小比 GAE 明显要大. 从表中结果还可以看出: 本文的 GKCI 方法相比已有方法, 在 K=10 时优势最为明显; 随着 K 的增大, 优势越来越小. 说明 GKCI 预测的前 10 个候选关键类中, 可识别更多真正的关键类.

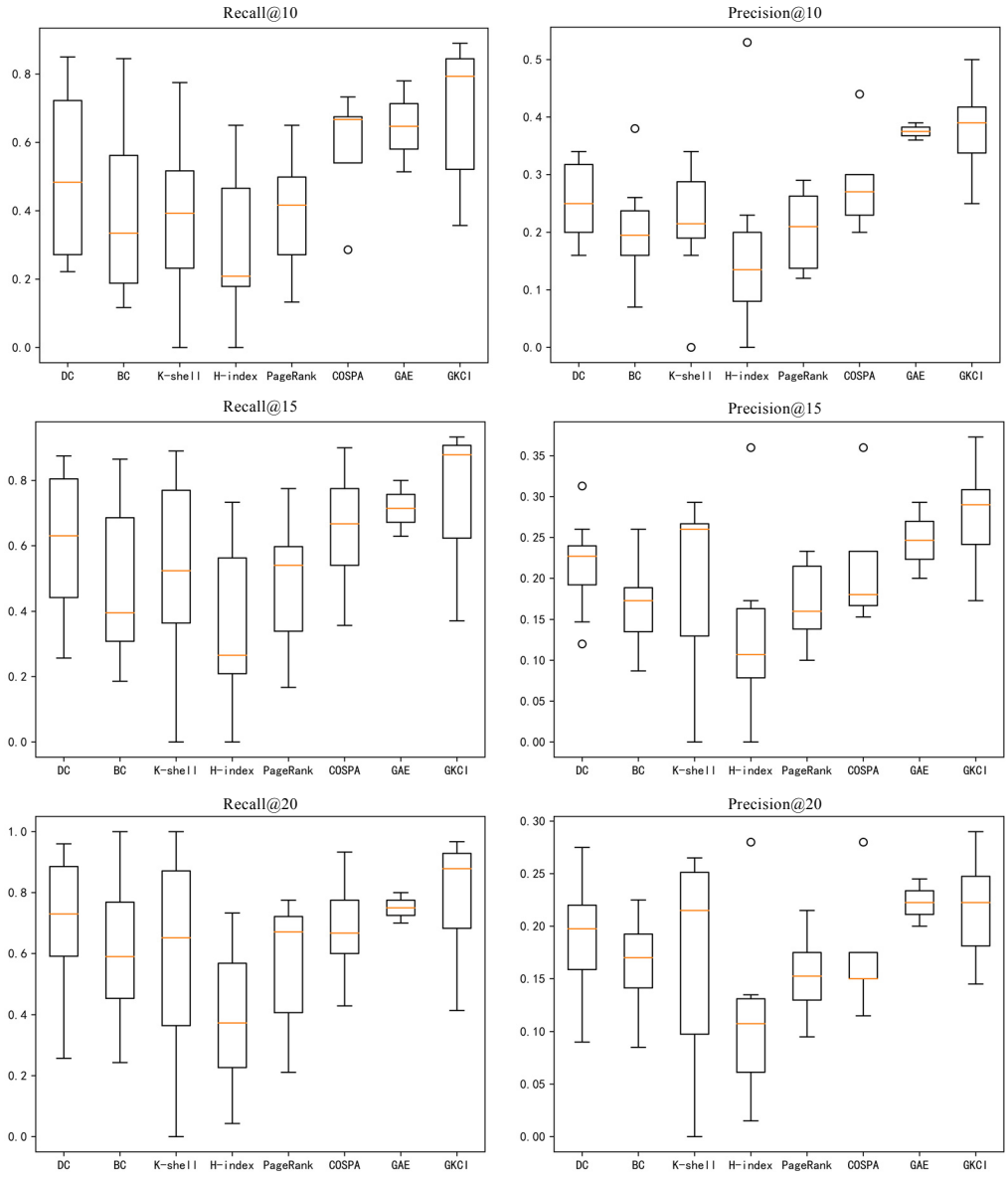


图 10　各种方法的召回率/精确率对比

表 6　GKCI 与各种方法的召回率对比结果

Method	Avg. (recall)			Avg. (GKCI-*) (%)			GKCI vs. * (Sig.p<0.05, d)		
	@10	@15	@20	@10	@15	@20	@10	@15	@20
DC	0.503	0.612	0.701	19.0	14.7	8.5	0.013 (**0.5**)	0.014 (0.469)	0.04 (0.25)
BC	0.405	0.482	0.617	28.8	27.7	16.9	0.011 (**0.656**)	0.004 (**0.656**)	0.016 (0.344)
K-shell	0.386	0.526	0.593	30.7	23.3	19.3	0.007 (**0.656**)	0.072 (**0.531**)	0.155 (0.344)
H-index	0.303	0.352	0.393	39.0	40.7	39.3	0.006 (**0.781**)	0.004 (**0.781**)	0.004 (**0.781**)
PageRank	0.391	0.486	0.572	30.2	27.3	21.4	1.03E-05 (**0.719**)	1.36E-04 (**0.656**)	0.001 (**0.609**)
COSPA	0.580	0.648	0.681	20.3	21.4	21.1	0.012 (**0.76**)	0.02 (**0.64**)	0.014 (**0.68**)
GAE	0.647	0.714	0.750	6.4	6.5	7.4	0.321 (**0.5**)	0.323 (**0.5**)	0.351 (**0.5**)
GKCI	0.693	0.759	0.786	–	–	–	–	–	–

注: Avg. (GKCI-*)表示 GKCI 相对于基准方法的提升幅度, Sig.p<0.05 和 d 分别是 Wilcoxon 符号秩检验和 Cliff' Delta, 其中, 下划线标记为差异显著, 粗体数值表示 GKCI 的影响大小差异显著

表 7 GKCI 与各种方法的准确率对比结果

Method	Avg. (precision)			Avg. (GKCI-*) (%)			GKCI vs. * (Sig.p<0.05, *d*)		
	@10	@15	@20	@10	@15	@20	@10	@15	@20
DC	0.256	0.217	0.191	12.0	5.9	2.5	0.008 (**0.75**)	0.01 (0.469)	0.033 (0.234)
BC	0.204	0.167	0.164	17.2	10.9	5.2	0.01 (**0.859**)	0.005 (**0.813**)	0.014 (**0.563**)
K-shell	0.218	0.202	0.174	15.8	7.4	4.2	0.001 (**0.828**)	0.027 (0.469)	0.123 (0.172)
H-index	0.173	0.131	0.111	20.3	14.5	10.5	0.01 (**0.75**)	0.006 (**0.766**)	0.007 (**0.781**)
PageRank	0.204	0.170	0.152	17.2	10.6	6.4	1.07E-04 (**0.859**)	0.002 (**0.813**)	0.008 (**0.688**)
COSPA	0.288	0.219	0.174	11.0	7.4	5.5	0.018 (**0.6**)	0.035 (**0.6**)	0.03 (**0.6**)
GAE	0.375	0.247	0.223	3.5	5.6	2.0	0.258 (**0.5**)	0.416 (**1.0**)	0.295 (**0.5**)
GKCI	0.376	0.276	0.216	–	–	–	–	–	–

注: 下划线和粗体数值与表 6 含义相同

5 讨 论

本节主要讨论本文涉及的一些实验处理问题、应用价值以及不足之处.

对于 RQ1, 实验结果表明, 考虑类之间依赖关系的权重比方向价值更大. 一种可能的解释是: GKCI 方法中对节点的前继邻居节点和后继邻居节点的分数均进行了聚合, 实际上等价于一定程度上弱化了依赖方向的影响. 但同时考虑以上两个因素时效果最佳, 即 GKCI 在有向加权的软件网络上关键类识别效果最好. 不同于普通的复杂网络, 类依赖网络更注重入度. 节点 v 如果发生了改变, 这种改变会因为级联关系的影响传递给其后继节点, 而对前继节点的影响不大. 也就是说, 对于目标节点, 其前继邻居节点的分数可能会比后继邻居节点的分数更为重要. 在未来的工作中, 本文会考虑在公式(10)中加入权重机制, 给前继邻居节点和后继邻居节点的分数分配不同的权重. 此外, 为了方便和统一, 本文输入到 Node2Vec 中学习的 CDN 全部忽略了方向和权重, 只关注节点的邻接结构信息.

本文采用 DependencyFinder 将软件系统解析为 XML 文件, 生成的 XML 文件中包含了本文所需要的类与类之间的交互关系, 并采用自主研发的解析程序进一步解析 XML 文件, 将类之间的依赖关系提取为软件网络图文件格式. 根据以往的工作[36,37], 本文只提取常用的 3 种依赖关系(聚合、继承、调用). 解析后的 XML 文件虽然包含多种依赖关系, 但并不能加以区分. 在实际应用场景中, 不同依赖关系的信息可能并不完全一致. 因此, 在未来的工作中, 我们会进一步考虑不同依赖关系的提取和区分.

对于 RQ2, 由于本文将数据集按 1:1 划分为训练集和测试集, 并比较各种方法在测试集中检索出关键类的性能, 为保证结果的真实性, 我们以作者论文中给出的数据集作对比, 其中, 前者使用了 6 个数据集(Ant 1.6.1、Argo UML 0.9.5、jEdit 5.1.0、jHotDraw 6.0b.1、jMeter 2.0.1 和 wro4j 1.6.3), 但由于在官网上并没有找到 Argo UML 0.9.5 版本数据, 所以只在剩余 5 个公共数据集上进行了对比; 而后者使用了 4 个数据集(Maven 4.5.0.2、Vuze 3.2.2、Ant 1.6.1 和 jMeter 2.0.1), 其中, Maven 4.5.0.2 和 Vuze 3.2.2 没有标注关键类信息, 因此只适合在 Ant 1.6.1 和 jMeter 2.0.1 两个数据集上作对比. COSPA 本质上是基于传统指标集成的方法, 实验结果表明: 本文的方法不仅比单个传统方法要好, 而且比这种集成方法也要好. 相比同样是基于 GNN 的 GAE 方法, 本文模型虽然在一定程度上也有所提升, 并且提升幅度的统计意义上不够显著, 但是影响大小 d 值仍在 0.5 及以上, 即表现为具有较大的影响差异.

最后, 本文也存在以下一些不足之处.

(1) 本文模型只采用一层简单的聚合架构也能取得较好的结果. 通过算法 1 可以看出, 本文模型的时间复杂度为 O(n), 其中, n 是类依赖网络中节点的数量. 根据经验, 聚合迭代层数越多, 则可以聚合更大邻域的分数. 而随着聚合层数的增多, 时间复杂度呈指数型增长, 即 O(n^k), 其中, k 为迭代层数. 在未来的工作中, 我们将本文模型扩展到更通用的架构, 扩展为包含多层分数的聚合层, 以检验这一改进是否能够提高关键类识别的准确率, 同时也能保证计算效率;

(2) 编程语言: 由于本文所选数据集均基于 Java 语言, 而实际场景中编程语言多种多样, 因此, 当本文模型扩展到其他编程语言, 如 C++或 Python 时, 结论可能会不成立. 在未来的工作中, 我们会将本

文的工作扩展到不同的编程语言中;

(3) 数据规模: 由于软件关键类识别领域中常用的公开数据集大部分都是小型系统, 文件数量在 1 000 以内, 而 Hibernate 的文件数量有 3 000 多个, GKCI 的预测性能在 Hibernate 不如其他几个项目. 但是其他几个基准模型在 Hibernate 的性能也远不如其他几个项目, 这可能是由于随着项目规模的扩大, 而关键类的数量始终保持在 10–20 个左右, 扩大了数据不平衡的问题, 使得模型很难在较少的候选关键类中预测出真正的关键类. 因此, 本文的结论在更大规模的开源项目上可能不成立. 在未来的工作中, 我们会选择大规模的软件系统或实际的应用工程来进一步验证本文方法.

6 总 结

本文针对软件系统中的关键类识别任务, 重点围绕软件网络建模中连边方向和权重的影响、图神经网络中目标节点的邻域聚合方式优化这两个方面, 在 8 个开源软件上进行了实证分析. 研究结果表明:

- 类依赖网络中考虑权重比方向更有助于关键类识别, 但在综合考虑有向加权情境下识别效果最好;
- 另外, 在对目标节点进行邻域聚合时, 聚合邻居节点的重要性得分比以往聚合邻居节点的表征向量效果更好;
- 最后, 本文所提 GKCI 方法相比基于复杂网络理论的经典方法占有绝对优势, 且对比两种最先进的同行方法, 我们的方法同样优势明显, 表现出在更少的候选类中可以预测出更多的真实关键类, 其中, 在返回的前 10 个候选关键类中, 召回率和精确率比最先进的方法分别提升了 6.4%和 3.5%.

致谢 在此, 向对本文实验过程中算法复现工作给予大力支持的潘伟丰老师团队表示真诚感谢, 并向对本文工作提出宝贵评审意见的审稿专家表示衷心的感谢.

References:

[1] LaToza TD, Venolia G, DeLine R. Maintaining mental models: A study of developer work habits. In: Proc. of the 28th Int'l Conf. on Software Engineering. 2006. 492–501.

[2] Maggie H, Katerina GP. Common trends in software fault and failure data. IEEE Trans. on software engineering. 2009, 35(4): 484–496.

[3] Wang S, Liu T, Nam J, *et al*. Deep semantic feature learning for software defect prediction. IEEE Trans. on Software Engineering, 2018.

[4] Li J, He P, Zhu J, *et al*. Software defect prediction via convolutional neural network. In: Proc. of the 2017 IEEE Int'l Conf. on Software Quality, Reliability and Security (QRS). IEEE, 2017. 318–328.

[5] Qu Y, Liu T, Chi J, *et al*. node2defect: Using network embedding to improve software defect prediction. In: Proc. of the 33rd IEEE/ACM Int'l Conf. on Automated Software Engineering (ASE). IEEE, 2018. 844–849.

[6] Ding Y, Li B, He P. An improved approach to identifying key classes in weighted software network. Mathematical Problems in Engineering, 2016.

[7] Jiang L, Jing Y, Hu S, *et al*. Identifying node importance in a complex network based on node bridging feature. Applied Sciences, 2018, 8(10): 1914.

[8] Zhang J, Luo Y. Degree centrality, betweenness centrality, and closeness centrality in social network. In: Proc. of the 2nd Int'l Conf. on Modelling, Simulation and Applied Mathematics (MSAM 2017). Atlantis Press, 2017. 300–303.

[9] Mahmoody A, Tsourakakis CE, Upfal E. Scalable betweenness centrality maximization via sampling. In: Proc. of the 22nd ACM SIGKDD Int'l Conf. on Knowledge Discovery and Data Mining. 2016. 1765–1773.

[10] Lü L, Zhou T, Zhang QM, *et al*. The H-index of a network node and its relation to degree and coreness. Nature Communications, 2016, 7(1): 1–7.

[11] Garas A, Schweitzer F, Havlin S. A *k*-shell decomposition method for weighted networks. New Journal of Physics, 2012, 14(8): 083030.

[12] Tortosa L, Vicent JF, Yeghikyan G. An algorithm for ranking the nodes of multiplex networks with data based on the PageRank concept. Applied Mathematics and Computation, 2021, 392: 125676.

[13] Pan WF, Li B, Ma YT, *et al*. Identifying the key packages using weighted PageRank algorithm. Acta Electonica Sinica, 2014, 42(11): 2174 (in Chinese with English abstract).

[14] Zeng C, Zhou CY, Lv SK, *et al*. GCN2defect: Graph convolutional networks for SMOTETomek-based software defect prediction. In: Proc. of the 32nd IEEE Int'l Symp. on Software Reliability Engineering (ISSRE). IEEE, 2021. 69−79.

[15] Gong M, Zhou H, Qin AK, *et al*. Self-paced co-training of graph neural networks for semi-supervised node classification. IEEE Trans. on Neural Networks and Learning Systems, 2022.

[16] Liu J, Zheng J, Wu J, *et al*. FA-GNN: Filter and augment graph neural networks for account classification in Ethereum. IEEE Trans. on Network Science and Engineering, 2022.

[17] Guo JY, Li RH, Zhang Y, Wang GR. Graph neural network based anomaly detection in dynamic networks. Ruan Jian Xue Bao/ Journal of Sofware, 2020, 31(3): 748−762 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/5903.htm [doi: 10.13328/j.cnki.jos.005903]

[18] Xie Y, Liang Y, Gong M, *et al*. Semisupervised graph neural networks for graph classification. IEEE Trans. on Cybernetics, 2022.

[19] Bai L, Cui L, Jiao Y, *et al*. Learning backtrackless aligned-spatial graph convolutional networks for graph classification. IEEE Trans. on Pattern Analysis and Machine Intelligence, 2020, 44(2): 783−798.

[20] Park N, Kan A, Dong XL, *et al*. Estimating node importance in knowledge graphs using graph neural networks. In: Proc. of the 25th ACM SIGKDD Int'l Conf. on Knowledge Discovery & Data Mining. 2019. 596−606.

[21] Grover A, Leskovec J. Node2Vec: Scalable feature learning for networks. In: Proc. of the 22nd ACM SIGKDD Int'l Conf. on Knowledge Discovery and Data Mining. 2016. 855−864.

[22] Veličković P, Cucurull G, Casanova A, *et al*. Graph attention networks. arXiv:1710.10903, 2017.

[23] Du X, Wang T, Pan W, *et al*. COSPA: Identifying key classes in object-oriented software using preference aggregation. IEEE Access, 2021, 9: 114767−114780.

[24] Zhang JX, Song K, He P, *et al*. Identification of key classes in software systems based on graph neural networks. Computer Science, 2021, 48(12): 149−158 (in Chinese with English abstract).

[25] Pan W, Ming H, Chang CK, *et al*. ElementRank: Ranking Java software classes and packages using a multilayer complex network-based approach. IEEE Trans. on Software Engineering, 2019, 47(10): 2272−2295.

[26] Pan W, Song B, Li K, *et al*. Identifying key classes in object-oriented software using generalized *k*-core decomposition. Future Generation Computer Systems, 2018, 81: 188−202.

[27] Şora I, Chirila CB. Finding key classes in object-oriented software systems by techniques based on static analysis. Information and Software Technology, 2019, 116: 106176.

[28] Li H, Wang T, Pan W, *et al*. Mining key classes in java projects by examining a very small number of classes: A complex network-based approach. IEEE Access, 2021, 9: 28076−28088.

[29] Young HP, Levenglick A. A consistent extension of Condorcet's election principle. SIAM Journal on Applied Mathematics, 1978, 35(2): 285−300.

[30] Young HP. Condorcet's theory of voting. American Political Science Review, 1988, 82(4): 1231−1244.

[31] Shah NB, Wainwright MJ. Simple, robust and optimal ranking from pairwise comparisons. The Journal of Machine Learning Research, 2017, 18(1): 7246−7283.

[32] Fan C, Zeng L, Ding Y, *et al*. Learning to identify high betweenness centrality nodes from scratch: A novel graph neural network approach. In: Proc. of the 28th ACM Int'l Conf. on Information and Knowledge Management. 2019. 559−568.

[33] Kitsak M, Gallos LK, Havlin S, *et al*. Identification of influential spreaders in complex networks. Nature physics, 2010, 6(11): 888−893.

[34] Wu Z, Pan S, Chen F, *et al*. A comprehensive survey on graph neural networks. IEEE Trans. on Neural Networks and Learning Systems, 2020, 32(1): 4−24.

[35] Hamilton W, Ying Z, Leskovec J. Inductive representation learning on large graphs. Advances in Neural Information Processing Systems, 2017, 30.

[36] He P, Li B, Ma Y, *et al*. Using software dependency to bug prediction. Mathematical Problems in Engineering, 2013.

[37] He P, Wang P, Li B, *et al*. An evolution analysis of software system based on multi-granularity software network. Acta Electonica Sinica, 2018, 46(2): 257 (in Chinese with English abstract).

[38] Macbeth G, Razumiejczyk E, Ledesma RD. Cliff's Delta calculator: A non-parametric effect size program for two groups of observations. Universitas Psychologica, 2011, 10(2): 545−555.

附中文参考文献:

[13] 潘伟丰, 李兵, 马于涛, 等. 基于加权 PageRank 算法的关键包识别方法. 电子学报, 2014, 42(11): 2174.

[17] 郭嘉琰, 李荣华, 张岩, 王国仁. 基于图神经网络的动态网络异常检测算法. 软件学报, 2020, 31(3): 748−762. http://www.jos.org.cn/1000-9825/5903.htm [doi: 10. 13328/j.cnki.jos.005903]

[24] 张健雄, 宋坤, 何鹏, 等. 基于图神经网络的软件系统中关键类的识别. 计算机科学, 2021, 48(12): 149−158.

[37] 何鹏, 王鹏, 李兵, 等. 基于多粒度软件网络模型的软件系统演化分析. 电子学报, 2018, 46(2): 11.

周纯英(1998－), 女, 硕士, 主要研究领域为复杂网络, 软件缺陷预测, 软件关键类识别, 图神经网络.

何鹏(1988－), 男, 博士, 副教授, CCF 专业会员, 主要研究领域为软件工程, 复杂网络, 软件度量, 软件缺陷预测.

曾诚(1976－), 男, 博士, 教授, CCF 专业会员, 主要研究领域为人工智能, 行业软件研发.

张龑(1973－), 男, 博士, 教授, 主要研究领域为信息安全, 代码测试, 教育大数据.