基于编译优化的软件缺陷预测研究

陈 勇[1], 徐 超[1], 何炎祥[2], 沈凡凡[1]
(1. 南京审计大学信息工程学院, 江苏南京 211815; 2. 武汉大学计算机学院, 湖北武汉 430072)

摘 要: 软件缺陷预测有助于提高软件质量,合理配置软件测试资源,目前已经有不少基于软件度量指标的缺陷预测模型. 然而,现有的软件度量指标主要集中在源代码的结构信息上,程序语义信息考虑较少. 编译优化是对程序语义进行深入分析的结果,直观地认为它应该在一定程度上能够反映程序的语义信息,有助于软件缺陷预测. 因此,为分析编译优化度量指标对软件缺陷预测的影响,本文首先基于当前编译器中广泛使用的优化选项,设计了 9 种编译优化度量指标. 结合源代码结构层面的度量指标,构建了 5 种软件缺陷预测度量模型. 利用 weka 中提供的 13 种常用的分类器,对比分析了添加不同优化度量指标的模型效果,对编译优化度量与软件缺陷预测之间的关系进行了评价,同时与 DP-CNN(Defect Prediction via Convolutional Neural Network)模型进行了对比. 实验结果表明:编译优化度量指标对软件缺陷预测的召回率有显著影响;在代码复杂度度量指标的基础上增加编译优化度量指标,可以提升所有软件缺陷预测模型的性能,平均提升幅度约为 5%;基于代码大小的优化度量和基于性能的优化度量具有各自的特点,两者相结合可以在软件缺陷预测中获得更好的性能.

关键词: 编译优化; 软件度量; 软件缺陷预测
中图分类号: TP391 **文献标识码:** A **文章编号:** 0372-2112 (2021)02-0216-09
电子学报 URL: http://www.ejournal.org.cn **DOI:** 10.12263/DZXB.20200607

The Research of Compilation Optimization on Software Defect Prediction

CHEN Yong[1], XU Chao[1], HE Yan-xiang[2], SHEN Fan-fan[1]
(1. *School of Information Engineering, Nanjing Audit University, Nanjing, Jiangsu* 211815, *China*;
2. *School of Computer Science, Wuhan University, Wuhan, Hubei* 430072, *China*)

Abstract: Software defect prediction helps improve software quality and allocate software test resources reasonably. Many defect prediction models based on software metrics have been proposed. However, the existing software metrics are mainly focused on structure information of source code, and the semantic information is lacking. Compilation optimization is the result of deep analysis of program semantics, and intuitively we believe that it should reflect the semantic information of the program in some ways to help defect prediction. Based on the optimization options widely used in the current compiler, this paper extracts 9 compilation optimization metrics, and proposes five types of metrics models that designed by different metrics sets. The relationship between compilation optimization metrics and software defect predictions was evaluated by 13 commonly used classifiers in weka, and also compared with DP-CNN. Experimental results show: Compilation optimization metrics have a significant impact on the recall rate of software defect prediction; Static code metrics combined with compilation optimization metrics can improve the performance of software defect prediction in all classifiers, which can improve the performance of prediction by about 5%; Code size based optimization metrics and performance based optimization metrics have their characteristics, combined both of them can get better performance in software defect prediction.

Key words: compilation optimization; software metrics; software defect prediction

---

收稿日期:2020-06-22;修回日期:2020-08-27;责任编辑:马兰英
基金项目:国家自然科学基金(No. 71972102, No. 61640220, No. 61902189);教育部人文社会科学研究规划基金(No. 19YJAZH100);南京审计大学青年教师科研培育项目(No. 19QNPY018);江苏省自然科学基金(No. BK20180821)

1　引言

在人们生活的各个方面,软件的比重越来越大,由软件缺陷引起的问题也越来越严重. 软件缺陷预测已经成为软件工程的重点研究领域,大量研究人员致力于开发精准高效的软件缺陷预测模型和工具,帮助软件工程师和测试人员快速定位软件中的缺陷,以提升软件的质量[1~6].

软件缺陷预测模型大多基于软件度量因子构建,因此,软件度量因子的优劣直接影响软件缺陷预测模型的有效性. 传统的软件度量因子主要可分为代码复杂度的度量因子(如 Halstead、McCabe[7]、CK[8] 等)和开发过程度量因子(如基于代码修改的度量[9]、基于项目团队组织构架的度量[10]、基于开发人员的度量[11]等)两大类. 这些度量因子虽然有力支撑了软件缺陷预测模型的构建,在项目内和跨项目软件缺陷预测方面都取得了一定的成果[12],但由于这些度量因子缺乏语义层面的信息,对于很多语义相关的缺陷难以实现有效预测.

近年来,随着深度学习技术的发展,一些研究者开始探讨基于深度神经网络自动提取程序语义度量因子的技术. Jian Li 等[13]基于程序的抽象语法树(ASTs),首先提取符号向量,然后通过字嵌入将符号向量编码为数字向量. 接着利用卷积神经网络(CNN),从编码的数字向量中自动学习程序的语义和结构特征. 最后,将学习到的特征与传统的度量因子相结合,实现对软件缺陷的预测. Song Wang 等[14] 利用一个深度信任网络(DBN),基于程序的ASTs和源代码更改中提取的标记向量,来自动学习语义特征. Deyu Chen 等[15]同样基于程序的 ASTs,使用双向长短期记忆(BiLSTM)神经网络从嵌入的标记向量中自动学习语义特征. 此外,Anh Viet Phan 等[16]首先根据程序的汇编指令构造控制流图,然后应用基于多层有向图的卷积神经网络(DGCNNs)学习程序的语义特征. 这些方法虽然能够自动提取程序的部分语义特征,但由于他们主要基于深度学习算法,需要对程序的 ASTs 树节点或汇编指令节点逐个进行迭代分析,时间开销比较大,另一方面,ASTs 通常保持了源代码的所有信息,其中包含的冗余信息比较大,而汇编节点与具体的体系架构相关,普适性受到一定的限制.

而对于程序语义,编译器在程序转换过程中是应用最为明显的. 特别是在编译优化阶段,所有的优化都是基于程序语义等价的原则上开展的. 如果两段程序语义上存在不同,那么编译优化的结果将或多或少存在不同. 表 1 展示了 2 个典型的示例. 其中,示例 1 截取自文献[16],示例 2 来源于 github 开源项目 redis①. 表格中第 2 列表示正确代码,第 3 列表示有缺陷代码.

在第一个示例中,正确代码和有缺陷代码仅在第 7 行有微小的区别. 有缺陷的代码错将"i + =1"写成了"i =1",这将会导致外面的 for 循环变成死循环. 对于这个缺陷,不会引起基于传统度量因子的缺陷预测模型的注意[16],但如果从编译优化的角度来看,有缺陷的代码 for 循环内 i 的值保持为 1,i 在循环内的赋值将是冗余的,所以整个 for 循环体就会简化为"for(;1 <= N;){sum ++ ;}". 此时我们从代码量上就可以很容易的发现这个缺陷.

表 1　示例代码

	正确的代码	有缺陷的代码

示例 1	1 int sumto N( )
	2 {
	3 int sum = 0;
	4 for(i = 0;i < = N;)
	5 {
	6 sum + = i;
	7 i + = 1;
	8 }
	9 return sum;
	10 }
	11 int main( )
	12 {
	13 int n = 4;
	14 printf("%d",sumto N(n));
	15 }
		1 int sumto N( )
		2 {
		3 int sum = 0;
		4 for(i = 0;i < = N;)
		5 {
		6 sum + = i;
		7 i = 1;
		8 }
		9 return sum;
		10 }
		11 int main( )
		12 {
		13 int n = 4;
		14 printf("%d",sumto N(n));
		15 }

① https://github.com/antirez/redis

续表

	正确的代码	有缺陷的代码

示例2	1 #include <stdio.h>
	2 #define RIO_FLAG_READ_ERROR (1 <<0)
	3 #define RIO_FLAG_WRITE_ERROR (1 <<1)
	4 typedef struct _rio {
	5 int cksum, flags;
	6 }rio;
	7 static inline int rioGetReadError(rio *r) {
	8 return (r->flags & \
	RIO_FLAG_READ_ERROR) ! =0;
	9 }
	10 static inline int rioGetWriteError(rio *r) {
	11 return (r->flags & \
	RIO_FLAG_WRITE_ERROR)! =0;
	12 }
	13 int test(rio * a1)
	14 {
	15 if(rioGetReadError(a1))
	16 printf("1");
	17 else if(rioGetWriteError(a1))
	18 printf("2");
	19 return a1->flags;
	20 }
		1 #include <stdio.h>
		2 #define RIO_FLAG_READ_ERROR (1 <<0)
		3 #define RIO_FLAG_WRITE_ERROR (1 <<1)
		4 typedef struct _rio {
		5 int cksum, flags;
		6 }rio;
		7 static inline int rioGetReadError(rio *r) {
		8 return (r->flags & \
		RIO_FLAG_READ_ERROR) ! =0;
		9 }
		10 static inline int rioGetWriteError(rio *r) {
		11 return (r->flags & \
		RIO_FLAG_READ_ERROR)! =0;
		12 }
		13 int test(rio * a1)
		14 {
		15 if(rioGetReadError(a1))
		16 printf("1");
		17 else if(rioGetWriteError(a1))
		18 printf("2");
		19 return a1->flags;
		20 }

在第2个示例中,正确代码和有缺陷代码之间的唯一区别是第11行.这种错误通常是程序员在编程时复制代码块造成的.由于这两段代码几乎相同,基于传统度量因子的缺陷预测模型找不到差异.即使是基于程序ASTs的自动语义特征提取,因为这些方法在映射"RIO_FLAG_READ_ERROR"这样的常量时,为了节省符号的数量,会把他们表示为一个统一的常数符号,因此,这类方法也是无法识别这样一类特征的.但如果基于编译优化来考虑,这两段代码之间的区别是显而易见的.因为rioGetReadError和rioGetWriteError是内联函数,而在错误代码中,它们的定义完全相同,因此在编译分析时,第15行和第17行中的条件将认为是完全相同的,所以在优化过程中,后面一个条件(即17~18行中的代码)将可以安全删除.从而也可以发现缺陷代码与正确代码之间的区别.

因此,本文将研究编译优化与软件缺陷预测之间的关系.首先,基于主流编译器的常用编译优化选项,设计9个基于编译优化的软件缺陷度量因子.然后,将其与传统的软件缺陷度量因子相结合,构建了5种缺陷度量模型;最后,利用weka[①]的分类器,构建13种软件缺陷预测模型,并加入文献[14]的DP-CNN模型,在两个开源项目上对不同的缺陷度量模型进行了评估.实验结果表明,加入了编译优化度量因子后,能有效提升软件缺陷预测的性能,特别在召回率上,效果更佳显著.

2 基于编译优化的软件缺陷度量

编译器的大部分优化通常是基于编译器内部的中间表示(IR)进行的,如GCC的RTL,LLVM的LLVM IR等.因此,本文设计的编译优化度量指标将基于编译器的IR来构建.

同时,为探索编译优化度量指标与软件缺陷预测之间的联系,本文结合传统的代码复杂度度量指标,构建了5种度量模型,并加入文献[14]的DP-CNN模型,在13种软件缺陷预测模型上进行评估.

2.1 编译优化度量指标

编译优化是编译器的重要组成部分,针对不同的软件需求,编译器提供了种类繁多的编译优化选项.但为了方便使用,编译器通常会将具有类似目标的多个优化方法封装成一个简单的选项,如面向性能优化的"O2"选项、面向代码大小优化的"Os"选项等.本文主要在宏观上探讨编译优化对软件缺陷预测的影响,因此我们将根据GCC、LLVM这些主流编译器的简化优化选项为基础,设计编译优化度量指标,具体包括9个方

① https://www.cs.waikato.ac.nz/ml/weka/

面,如表 2 所示.

表 2　编译优化度量指标

度量指标	描述
O0_LOC	无优化情况下,IR 指令条数
O1_LOC	“-O1”优化选项下,IR 指令条数
O2_LOC	“-O2”优化选项下,IR 指令条数
O3_LOC	“-O3”优化选项下,IR 指令条数
Os_LOC	“-Os”优化选项下,IR 指令条数
O1_OPRATE	“-O1”优化选项下的代码长度优化率,即:O1_LOC/O0_LOC
O2_OPRATE	“-O2”优化选项下的代码长度优化率,即:O2_LOC/O0_LOC
O3_OPRATE	“-O3”优化选项下的代码长度优化率,即:O3_LOC/O0_LOC
Os_OPRATE	“-Os”优化选项下的代码长度优化率,即:Os_LOC/O0_LOC

其中,O0_LOC 是基准指标,Os_LOC、Os_OPRATE 是代码大小优化相关的指标,其余指标是性能优化的相关指标.

对于以上 9 个编译优化度量指标,他们只与编译器内部的 IR 指令条数相关,而在源程序编译的过程中,主流编译器都提供了输出 IR 的选项,比如在 LLVM 编译框架中,只需在编译参数中添加“-emit-llvm”选项就可以输出 IR 代码①. 因此,为获得这 9 个编译优化度量指标,我们可以依次使用这些优化级别选项,编译源代码,生成对应的 IR 代码,然后计算 IR 代码的指令条数即可. 具体如算法 1 所示. 其中,输入的 SC 为源代码,Ox 为优化选项(即“O0”,“O1”,“O2”,“O3”,“Os”).

算法 1　编译优化因子提取算法

```
输入:源代码 SC;优化选项 Ox
输出: IR 指令的条数 count
Step1:获得 IR
  IR = clang < Ox > -emit-llvm < SC >
Step2:遍历 IR,统计 IR 指令数
  count = 0
  foreach i in IR
    if i is instruction then
      count ++
    endif
  endfor
```

2.2　基于编译优化的度量模型

为了评估编译优化度量指标与软件缺陷预测之间的关系,本文结合传统的基于代码复杂度的度量指标,构建了 5 种度量模型.

(1) 代码复杂度度量模型(CM):该模型是基准度量模型,仅由传统的基于代码复杂度的度量指标构成.

(2) 优化指标度量模型(OM):该模型仅由本文提出的 9 个编译优化度量指标构成,用于直接评估优化指标对软件缺陷预测的影响.

(3) 混合模型(HM):该模型包含 CM 和 OM 所有度量指标,用于评估编译优化度量指标的加入是否会提升软件缺陷预测的总体性能.

为了评估性能相关的优化与代码大小相关的优化对软件缺陷预测的影响,本文还增加了基于性能优化的混合模型和基于代码大小优化相关的混合模型.

(4) 基于性能优化的混合模型(POHM):该模型主要用于衡量性能优化对软件缺陷预测的影响. 因为在主流编译器中,“Os”优化通常用于指定编译器主要进行代码大小相关的优化,因此该模型在 HM 模型的基础上,除去了与 Os 的“Os_LOC”和“Os_OPRATE”指标.

(5) 基于代码大小优化的混合模型(SOHM):该模型主要用于衡量代码大小优化对软件缺陷预测的影响. 包含 CM 模型的全部度量指标,以及 OM 模型中“O0_LOC”、“Os_LOC”和“Os_OPRATE”三个指标. 之所以不再使用“O1”、“O2”、“O3”这三个优化选项对应的指标,是因为这三个优化选项对应的优化主要以程序性能优化为主.

其中,CM 度量指标主要利用 SourceMonitor② 工具从源代码中提取,包括 12 个维度的信息,如表 3 所示.

表 3　基于代码复杂度的度量指标

度量指标	描述
Lines	源文件的总行数
Statements	可执行语句的行数(除去空行和注释)
% Branch	分支语句的数目
% Comments	注释的行数
ClassesDefs	类定义的数目
Methods/Class	每个类中平均定义的方法数目
Avg Stmts/Method	每个方法中平均定义的可执行语句数
Max Complexity	最复杂的方法定义的语句数
Max Depth	层次最深的基本款定义的语句数
Avg Depth	基本块的平均深度
Avg Complexity	平均复杂度
Functions	总的方法数目

2.3　软件缺陷预测模型

为验证优化指标对于不同软件缺陷预测模型的影

① https://llvm.org

② http://www.campwoodsw.com/sourcemonitor.html

响,本文基于 weka 的分类模型,构建了 13 种软件缺陷预测模型,其功能如表 4 所示.

表 4 基于 weka 分类器的软件缺陷预测模型

Weka 中的分类模型	描述
BayesNet	贝叶斯网络分类器
NaiveBayes	基于估计器的朴素贝叶斯分类器
NaiveBayesUpdateable	可更新的多项式朴素贝叶斯分类器
MultilayerPerceptron	基于反向传播学习的多层感知机分类器
IBk	K 近邻分类器
KStar	K * 分类器
AdaBoostM1	Adaboost M1 分类器
RandomizableFiltered Classifier	基于随机过滤器的分类器
PART	基于 PART 决策表分类器
HoeffdingTree	基于霍夫曼树的分类器
J48	C4.5 决策树分类器
RandomForest	随机森林分类器
RandomTree	随机树分类器

3 实验结果与分析

3.1 数据集

在本文中,我们使用 bitcoin① 和 Tesseract-OCR② 这两个开源项目作为我们的验证数据集. 其中,bitcoin项目是第一个基于区块链技术的数字货币开源实现,它的代码缺陷将会直接导致用户的经济损失. Tesseract-OCR 项目是一个免费的开源且具有高精准率的 OCR 图像识别引擎,是目前应用最广泛的文本识别技术之一.

```
for f in $(find ~/benchmark/<PROJECTNAME> -maxdepth 3
    -name "*.[cpp|c]" ) ; do
    echo -n $f >> ../fixNum.txt
    echo -n ":" >> ../fixNum.txt
    git log --grep='[F|f][i|I][X|x]' --after="<ReleaseTime>"
        --pretty=oneline $f|sed -n '$=' >>../fixNum.txt
    echo "" >>../fixNum.txt
done
```

图1 源代码修复信息提取脚本

本文主要从文件级评估软件缺陷,所以为了收集这两个开源项目的缺陷,本文首先使用 git 命令获取这两个项目的正确版本源代码. 然后,执行图 1 所示的 shell 脚本. 其中“ < PROJECTNAME > ”即项目的名称,“ < ReleaseTime > ”即提取的对应版本发布的时间. 由于自动的缺陷信息提取技术会产生标记偏差[17],因此本文在获得所有的修复信息以后,采用手工方式进行缺陷信息标记. 最终获得的数据集如表 5 所示.

表 5 数据集相关信息

数据集	版本	时间周期	文件数	缺陷数
Bitcoin	v0.16.0	2018.2.23 – 2020.1.31	246	74(30.1%)
Tesseract-ocr	3.05.00	2017.2.17 – 2020.1.31	233	112(48.1%)

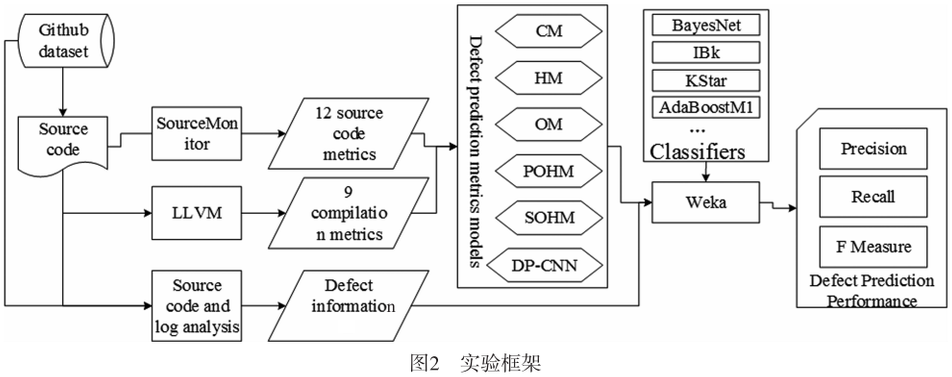

图2 实验框架

3.2 实验框架

为评估编译优化指标与软件缺陷预测之间的关系,本文构建了如图 2 所示的实验框架.

(1) 从 Github 上获得这两个项目的源代码后,利用 SourceMonit 从源代码中提取出 12 个维度的 CM 度量指标. 对于 9 个编译优化指标,我们主要基于 LLVM 进行提取. 同时结合 log 信息,标记源代码中有缺陷的文件.

(2) 利用提取的 CM 度量指标以及 9 个编译优化度量指标,构建 5 种度量模型. 此外,为与卷积神经网络自动提取的特征进行对比,本文还加入了文献[14]提出的 DP-CNN 模型.

(3) 借助 Weka,实现了 13 种不同软件缺陷预测模型,对不同度量模型下软件缺陷预测能力进行评估.

3.3 实验结果分析

在具体的评估过程中,本文采用召回率(Recall)、准确率(Precision)和 *F*1 值来评估模型的预测能力. 这些评估指标主要基于表 6 所示的混淆矩阵.

① https://github.com/bitcoin/bitcoin

② https://github.com/tesseract-ocr

表 6　预测结果混淆矩阵

	预测为有缺陷	预测为无缺陷
真实有缺陷	*TP*	*FP*
真实无缺陷	*FN*	*TN*

召回率、准确率和 $F1$ 值这三个指标的定义如下:

$$\text{Recall} = \frac{TP}{TP + FN}, \quad \text{Precision} = \frac{TP}{TP + FP}$$

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

基于以上三个指标,本文以 CM 模型为基准,根据以下公式计算其他模型相对于 CM 模型的提升水平.

$$H_i^K = \frac{K_i - CM_i}{CM_i};\ i \in \{\text{Recall}, \text{Recision}, F1\}$$

$$K \in \{\text{HM}, \text{OM}, \text{POHM}, \text{SOHM}, \text{DP-CNN}\}$$

其中,K_i 表示度量模型 K 评估指标 i 的值,CM_i 表示 CM 度量模型评估指标 i 的值.

表 7 和表 8 分别显示了 5 种不同的度量模型在 13 种缺陷预测模型中相对于 CM 模型在查准率和查全率上的提升率. 这 13 种缺陷预测模型均使用了 Weka 的默认参数. 为了避免数据分布对实验的影响,本文主要使用十折交叉验证法,并重复做了 10 次实验,然后将这 10 次结果的平均值作为实际的评估值.

表 7　不同度量模型在不同软件缺陷预测模型下召回率的增长率

Models	Tesseract-OCR					Bitcoin				
	$H_{\text{Recall}}^{\text{HM}}$	$H_{\text{Recall}}^{\text{OM}}$	$H_{\text{Recall}}^{\text{POHM}}$	$H_{\text{Recall}}^{\text{SOHM}}$	$H_{\text{Recall}}^{\text{DP-CNN}}$	$H_{\text{Recall}}^{\text{HM}}$	$H_{\text{Recall}}^{\text{OM}}$	$H_{\text{Recall}}^{\text{POHM}}$	$H_{\text{Recall}}^{\text{SOHM}}$	$H_{\text{Recall}}^{\text{DP-CNN}}$
BayesNet	0.03%	3.80%	0.03%	0.03%	0.03%	5.77%	11.53%	6.00%	5.54%	3.53%
NaiveBayes	11.60%	16.97%	11.37%	1.34%	7.66%	5.52%	9.12%	3.60%	5.76%	3.71%
NaiveBayesUpdateable	11.86%	17.24%	11.63%	1.58%	9.09%	6.69%	17.50%	7.85%	4.37%	3.60%
MultilayerPerceptron	0.84%	22.22%	0.28%	0.26%	1.35%	3.17%	31.03%	3.64%	4.75%	10.03%
IBk	9.86%	−8.05%	9.02%	8.16%	8.98%	5.32%	−9.79%	3.78%	6.44%	3.30%
KStar	18.21%	−7.61%	7.34%	16.75%	12.19%	4.32%	−12.66%	2.01%	2.93%	8.36%
AdaBoostM1	7.26%	6.00%	0.19%	7.13%	4.65%	5.15%	14.42%	3.61%	3.86%	3.49%
RandomizableFilteredClassifier	1.99%	−11.46%	5.13%	2.83%	1.56%	7.00%	−11.66%	5.33%	4.33%	8.69%
PART	6.96%	47.95%	6.85%	4.16%	9.83%	14.82%	26.19%	12.06%	11.72%	5.17%
HoeffdingTree	11.46%	24.08%	13.36%	0.29%	15.46%	2.98%	19.88%	3.48%	0.99%	3.97%
J48	1.75%	42.97%	0.35%	2.76%	4.27%	10.71%	17.74%	11.21%	8.70%	9.13%
RandomForest	2.78%	−13.63%	1.29%	1.63%	1.87%	1.67%	8.11%	1.19%	2.03%	2.79%
RandomTree	1.34%	−11.52%	0.65%	2.53%	7.03%	8.62%	21.21%	9.45%	5.30%	8.45%
Average	6.61%	9.92%	5.19%	3.80%	6.46%	6.29%	10.97%	5.63%	5.13%	5.71%

由表 7 可以看出,在召回率方面,编译优化指标构成的度量模型 OM 与代码复杂度度量模型 CM 各具特点,对于某些软件缺陷预测模型,如 MultilayerPerceptron、PART 等模型,OM 明显优于 CM 模型,其召回率比 CM 模型提升了 20% 以上. 特别是在 Tesseract-OCR 项目的 PART 预测模型中,OM 的召回率更是比 CM 提升了 47.95%. 但在某些软件缺陷预测模型中,CM 优于 OM,如 IBk、KStar 等,这可能是由于基于不同分类算法构建的预测模型,在特征提取时所使用的方式不同而引起的. 但总的来说,在这 13 种缺陷预测模型中,OM 的平均召回率是最高的,比 CM 提升幅度可达 10% 左右,相对于 DP-CNN 模型,也提升了 4% 左右. 同时,通过对比 HM、POHM、SOHM 和 CM 数据,我们也可以看出:由于在 CM 模型的基础上加入了全部或者部分的编译优化度量指标,对于所有的软件缺陷预测模型,HM、POHM 和 SOHM 模型的召回率都优于单纯的代码复杂度度量模型 CM. 平均来说,HM 的召回率比 CM 提升幅度超过 6% 以上. 因此,编译优化指标对于提升软件缺陷预测的召回率有很强的支撑作用,加入编译优化度量指标有利于发现更多的软件缺陷.

在缺陷预测的准确率方面,由表 8 可以看出,对于 OM 度量模型,由于缺乏代码复杂度信息,在很多不少预测模型中,软件预测的准确率都相对于 CM 度量模型有所下降,如 BayesNet、AdaBoostM1、J48、RandomForest 等,OM 的平均准确率,两个测试项目均要略低于 CM 度量模型. 但对于 HM 度量模型,由于结合 CM 和编译优化度量指标,所以预测准确率较高,在所有软件缺陷预测模型中,其值均略优于 CM 模型. 特别对于 KStar 软件缺陷预测模型,基于 HM 的软件缺陷预测准确率提升了 10% 以上. 对于所有缺陷预测模型而言,在这两个测试项目中,HM 的准确率比 CM 平均都提升了 3% 以上.

表 8 不同度量模型在不同软件缺陷预测模型下准确率

Models	Tesseract-OCR					Bitcoin				
	$H^{HM}_{Precision}$	$H^{OM}_{Precision}$	$H^{POHM}_{Precision}$	$H^{SOHM}_{Precision}$	$H^{DP\text{-}CNN}_{Precision}$	$H^{HM}_{Precision}$	$H^{OM}_{Precision}$	$H^{POHM}_{Precision}$	$H^{SOHM}_{Precision}$	$H^{DP\text{-}CNN}_{Precision}$
BayesNet	1.54%	1.17%	1.17%	1.25%	1.05%	0.12%	-10.27%	-1.48%	-3.59%	0.80%
NaiveBayes	1.51%	-1.06%	-0.27%	0.00%	1.34%	3.24%	-2.49%	1.00%	0.75%	0.31%
NaiveBayesUpdateable	1.51%	-1.06%	-0.27%	0.00%	1.45%	3.21%	0.98%	2.65%	3.21%	2.68%
MultilayerPerceptron	4.60%	4.10%	4.84%	4.00%	7.03%	2.95%	-4.85%	2.43%	2.25%	3.15%
IBk	9.07%	0.90%	6.98%	5.70%	6.50%	6.66%	1.36%	5.44%	5.98%	3.01%
KStar	17.00%	6.73%	14.59%	10.77%	9.89%	12.70%	9.17%	11.29%	11.82%	9.15%
AdaBoostM1	3.04%	-17.66%	-1.42%	3.12%	0.09%	2.49%	-2.49%	2.11%	1.00%	4.11%
RandomizableFilteredClassifier	8.39%	-3.00%	11.63%	7.77%	6.66%	1.25%	-6.79%	0.18%	0.00%	6.20%
PART	1.04%	-4.42%	1.40%	-0.42%	3.61%	2.71%	-3.56%	1.53%	0.17%	3.02%
HoeffdingTree	0.93%	0.90%	1.52%	-3.28%	3.38%	5.13%	0.63%	6.63%	1.00%	1.21%
J48	0.25%	-4.36%	0.32%	0.38%	0.63%	0.17%	-7.08%	-3.88%	-2.87%	1.70%
RandomForest	0.50%	-12.08%	-0.11%	0.74%	1.58%	0.13%	-2.15%	0.00%	0.13%	1.10%
RandomTree	4.12%	-4.34%	5.87%	0.88%	1.25%	1.30%	-2.96%	1.30%	5.00%	5.78%
Average	4.11%	-2.63%	3.56%	2.38%	3.42%	3.24%	-2.35%	2.25%	1.91%	3.25%

为进一步综合评估编译优化指标对不同软件缺陷预测模型效果的影响,本文以 CM 为基准,计算了这两个项目中 HM、OM、POHM、SOHM、DP-CNN 在 *F*1 值的提升率,如图 3、图 4 所示.

从图 3 和图 4 可以看出,OM 由于缺乏代码复杂度相关指标的支撑,它受软件缺陷预测模型和具体的缺陷影响较大,总的评估值波动较大. 比如在 AdaBoostM1 缺陷预测模型中,在 Tesseract-OCR 项目中,它的 *F*1 值优于 CM,其提升值为正,但在 Bitcoin 项目中,它的 *F*1 值低于 CM,其提升值为负. 而结合了代码复杂度相关指标后,对于所有软件缺陷预测模型,HM、POHM 和 SOHM 的 *F*1 值均优于 CM,HM 的 *F*1 值平均提升率接近 5%. POHM 和 SOHM 在不同的软件缺陷预测模型中表现各具特点,如 Tesseract-OCR 项目中,POHM 在 NativeBayes 中综合表现明显优于 SOHM,但在 Bitcoin 项目中,POHM 在 NativeBayes 中的综合表现又略低于 SOHM,这可能与具体的软件缺陷与特征的敏感程度相关. 但总的来说,POHM 和 SOHM 均优于 CM,且他们两者中综合评估最好的值基本与 HM 相当. 因此,为了达到更好的软件缺陷预测效果,建议结合性能优化度量指标和代码大小优化指标进行软件缺陷度量.

同时,通过对比 HM 模型和 DP-CNN 模型的召回率、准确率和 *F*1 值,我们可以看出,HM 模型与 DP-CNN 模型在不同的缺陷预测模型上表现各不相同,这可能是由于两者提取的特征方式的不同而引起的. HM 模型主要是基于统计的方法进行的提取,所以在类似的模型如 NaiveBayes、NaiveBayesUpdateable 等上面表现更好,而 DP-CNN 是基于神经网络自动提取的特征,所以其在基于感知机的 MultilayerPerceptron 方法上表现更好,同时,他们都结合了传统的度量指标,所以相对来说表现都较传统方法有所提升. 总的来说,在这 13 种缺陷预测模型中,HM 模型与 DP-CNN 模型的平均 *F*1 值

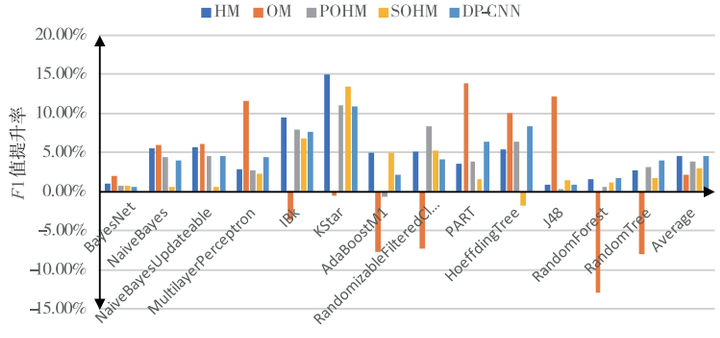

图3 Tesseract-OCR项目不同软件缺陷预测模型*F*1值提升率

相差不大,因此可以说明两个度量模型的应用的普适性基本相当.

此外,我们也可以看到,度量指标并非越多越好,比如 HM、POHM 和 SOHM 的召回率,在某些预测模型中(如 PART 等),甚至远低于 OM 模型,这也在侧面验证了维度灾难对预测模型的负面影响.

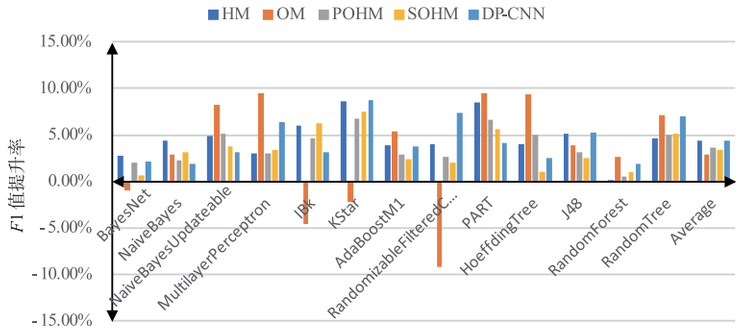

图4　Bitcoin项目不同软件缺陷预测模型*F*1值提升率

4　结论

为了探讨编译优化与软件缺陷预测之间的关系,本文以编译器中间表示 IR 为基础,设计了 9 个基于编译优化的软件度量指标,并结合已有的代码复杂度指标,构建了 5 种软件缺陷度量指标模型. 在两个开源项目数据集上使用 13 种常用分类器软件缺陷预测模型进行对比实验,同时也与文献[14]中的 DP-CNN 模型进行了对比分析.

实验结果表明编译优化度量指标对软件缺陷预测的影响具有普适性,它的加入对各类软件缺陷预测模型都有积极作用,能够提升软件缺陷预测的综合性能,与基于卷积神经网络的 DP-CNN 模型基本相当. 尤其在召回率上,比单纯的代码复杂度度量指标最高提升幅度可达 47.95%. 同时,基于代码大小优化的度量指标和基于性能优化的度量指标各具特点,两者相结合可以在软件缺陷预测中获得更好的性能. 在后续工作中,我们将收集更多的开源项目来验证编译优化度量指标在软件缺陷预测中的通用性,同时进一步考虑如何对各优化指标进一步细分,以从更细的维度研究编译优化与软件缺陷预测之间的联系.

参考文献

[1] Duksan R, Jong I J, Jongmoon B. A transfer cost-sensitive boosting approach for cross-project defect prediction[J]. Software Quality Journal, 2017, 25(1): 235 – 272.

[2] Chakkrit T, Shane M, Ahmed E H, et al. An empirical comparison of model validation techniques for defect prediction models[J]. IEEE Transactions on Software Engineering, 2017, 43(1): 1 – 18.

[3] Zhang F, Ahmed E H, Shane M, et al. The use of summation to aggregate software metrics hinders the performance of defect prediction models[J]. IEEE Transactions on Software Engineering, 2017, 43(5): 476 – 491.

[4] Zaheed M, David B, Tracy H, et al. Reproducibility and replicability of software defect prediction studies[J]. Information & Software Technology, 2018, 99(7): 148 – 163.

[5] Chakkrit T, Shane M, Ahmed E H, et al. The impact of automated parameter optimization on defect prediction models [J]. IEEE Transactions on Software Engineering, 2019, 45(7): 683 – 711.

[6] Toshiki M, Naoshi U. Balancing the trade-off between accuracy and interpretability in software defect prediction [J]. Empirical Software Engineering, 2019, 24(4): 779 – 825.

[7] Thomas J M. A complexity measure[J]. IEEE Transactions on Software Engineering, 1977, SE2(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] D'Ambros M, Lanza M, Robbes R. An extensive comparison of bug prediction approaches[A]. Proceedings of the 7th Working Conference on Mining Software Repositories [C]. Cape Town: IEEE, 2010. 31 – 41.

[10] Mockus A. Organizational volatility and its effects on software defects[A]. Proceedings of the 18th International Symposium on Foundations of Software Engineering[C]. Santa Fe: ACM, 2010. 117 – 126.

[11] Jiang T, Tan L, Kim S. Personalized defect prediction [A]. Proceedings of the 28th International Conference on Automated Software Engineering[C]. Melbourne: IEEE, 2013. 279 – 289.

[12] 陈翔,赵英全,顾庆,等. 基于文件粒度的多目标软件缺陷预测方法实证研究[J]. 软件学报, 2019, 30(12): 3694 – 3713.
Chen X, Zhao YQ, Gu Q, et al. Empirical studies on multi-objective file-level software defect prediction method[J].

Journal of Software,2019,30(12):3694 – 3713. (in Chinese)
[13] Li J, He P, Zhu J, et al. Software defect prediction via convolutional neural network[A]. 2017 IEEE International Conference on Software Quality, Reliability and Security[C]. Prague:IEEE,2017. 318 – 328.
[14] Wang S, Liu T, Nam J, et al. Automatically learning semantic features for defect prediction[A]. 2016 IEEE/ACM 38th International Conference on Software Engineering(ICSE)[C]. Austin:IEEE,2016. 297 – 308.
[15] Chen D, Chen X, Li H, et al, DeepCPDP: Deep learning based cross-project defect prediction[J]. IEEE Access, 2019,7:184832 – 184848.
[16] Phan A V, Nguyen M L, Bui L T, et al. Convolutional neural networks over control flow graphs for software defect prediction[A]. 2017 IEEE 29th International Conference on Tools with Artificial Intelligence [C]. Boston: IEEE,2017. 45 – 52
[17] Bird C, Bachmann A, Aune E, et al. Fair and balanced? Bias in bug fix datasets[A]. Proceedings of the 7th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT International Symposium on Foundations of Software Engineering[C]. Amsterdam, Netherlands:ACM,2009. 24 – 28.

作者简介

陈 勇 男,1986 年出生,湖南娄底人. 博士、高级工程师,研究方向为软件可靠性、编译优化、大数据审计、嵌入式系统优化.
E-mail:chenyong@ nau. edu. cn

徐 超(通信作者) 男,1980 年出生,湖北红安人. 博士、教授,研究方向为软件可靠性、大数据审计、区块链技术与应用.
E-mail:xuchao@ nau. edu. cn