2018 年 10 月 第 10 期
Vol. 39 No. 10 2018

一种半监督集成学习软件缺陷预测方法

张 肖,王黎明

(郑州大学 信息工程学院,郑州 450001)
E-mail: xz_zhangxiao@163.com

摘 要: 针对软件缺陷预测中标记样本难以获取以及分类不平衡的问题,提出一种基于半监督集成学习方法的软件缺陷预测模型(Tri_Adaboost). 一方面利用欠采样方法以及半监督学习对标记样本进行扩充,随机选取一部分无标记样本进行预标注,缓解标记样本不足的问题;另一方面,利用 SMOTE 方法对扩充后的标记样本进行采样,然后使用 AdaBoost 集成方法对标记样本集进行预测. 本文在 NASA MDP 数据集及基于开源项目下生成的空指针引用缺陷数据集上,验证模型的有效性,较于四种基本的机器学习分类方法,Tri_Adaboost 算法在 F-measure 和 AUC 上均能取得较高的值.

关 键 词: 软件缺陷预测;分类不平衡;半监督学习;AdaBoost
中图分类号: TP391 **文献标识码:** A **文 章 编 号:** 1000-1220(2018)10-2138-08

Semi-supervised Ensemble Learning Approach for Software Defect Prediction

ZHANG Xiao, WANG Li-ming
(School of Information Engineering, Zhengzhou University, Zhengzhou 450001, China)

Abstract: Aiming at the problem that the large number of labeled samples in the software defect prediction are difficult to obtain and the existence of class imbalanced in the software system, a semi-supervised ensemble learning method is proposed. On the one hand, under-sampling method and semi-supervised learning are used to extend the labeled samples, some unlabeled samples are randomly selected for pre-labeled to alleviate the insufficient of labeled samples; On the other hand, the SMOTE method is used to sample the extended labeled samples, and then the AdaBoost ensemble method is used to predict the labeled sample set. The paper verifies the validity of the model based on the NASA MDP data set and the null pointer defect dataset generated under the open source project, compared with the four basic machine learning classification methods, Tri_Adaboost algorithm can achieve higher values on F-measure and AUC.

Key words: software defect prediction; class imbalance; semi-supervised learning; AdaBoost

1 引 言

随着社会的发展,软件的多样性以及普遍性已经渗入到人们的生活中,软件的可靠性是人们进行日常交易的重要保证,导致软件系统不可靠的主要原因之一是软件缺陷[1]. 软件缺陷预测分为动态缺陷预测和静态缺陷预测,静态软件缺陷预测是指在不运行程序的情况下,根据软件的历史开发数据以及已发现的缺陷,借助机器学习等方法来预测软件系统中的缺陷数目和类型等[2].

在软件缺陷预测中,标记样本难以获取,无标记样本数量较多且容易收集. 半监督学习利用标记样本训练模型,同时利用未标记样本加强学习过程,不仅提高了分类器的性能,而且使标记样本匮乏的问题得以缓解. 在软件测试中,80% 的缺陷集中分布于 20% 的模块内,因此软件缺陷数据集是高度类不平衡的[3]. 集成方法[4]、代价敏感方法[5] 以及抽样方法[6] 是解决分类不平衡问题的有效途径.

为了解决标记样本较少以及分类不平衡问题对预测模型造成的影响,提出一种基于半监督集成学习算法的软件缺陷预测模型. 将半监督 Tri_training 算法应用到集成学习算法 AdaBoost 中对缺陷数据集进行分类预测. 预测算法分为两个阶段:

1) 利用 Tri_training 算法以及欠采样方法对无标记样本进行标记,将新标记的样本添加到已标记样本中形成标记样本集.

2) 利用 SMOTE 算法对新的标记样本集进行采样,再使用 AdaBoost 集成学习方法对采样后的标记样本集进行预测.

在第一阶段中,半监督算法可以充分利用无标记样本解决标记样本不足的问题,在第二阶段中,集成算法能更好的解决分类不平衡的问题,并且能取得较好的预测效果.

2 相关工作

半监督学习能够充分利用少量的有标记样本和大量的未标记样本提高分类模型的性能,而集成学习通过对几个分类器的组合能够有效提高分类器的泛华性能,同时也能有效解决分类不平衡的问题. 因此,很多半监督学习和集成学习方法

收稿日期: 2017-12-04 收修改稿日期: 2018-02-06 作者简介: 张 肖,女,1992 年生,硕士研究生,研究方向为软件工程、数据挖掘等;王黎明(通信作者),男,1963 年生,博士,教授,研究方向为现代软件工程技术、分布式人工智能、数据挖掘等.

已经被广泛应用于软件缺陷预测领域.

2.1　半监督学习

文献 [7] 对 Tri-training 算法进行了改进,将随机下采样方法与 Tri-training 进行结合,有效的降低了分类不平衡和标记样本不足对预测模型性能的影响. 文献 [8] 等人研究利用了朴素贝叶斯方法建立了半监督缺陷预测模型,其结果表明朴素贝叶斯算法是构建小规模数据集半监督缺陷预测模型的最佳选择,提出的两阶段 YATSI 方法可以提高朴素贝叶斯在大规模数据集上的性能. 文献 [9] 提出了一种迭代的半监督方法 FTF,首先使用模型为所有的未标记实例设置标签,以确保样本中所有的实例都带有标签,然后在整个数据集上实现经典的监督过程. 结果表明,与传统的监督方法相比,FTF 有比较明显的优点. 文献 [10] 利用期望最大化(Expectation Maximization,EM)算法对半监督学习进行研究,使用有限的缺陷倾向性数据评估软件质量. 实验表明,基于 EM 的半监督分类提高了软件缺陷预测模型的泛化性能. 此外,半监督软件缺陷预测模型通常会产生比利用已知故障倾向性模块训练的决策树模型更好的性能.

2.2　集成学习

集成学习方法通过构建并结合多个分类器进行训练和分类,其泛化性能通常会优于单个分类器的性能. 文献 [11] 提出一种基于软件度量元的集成 KNN 软件缺陷预测方法,实验结果表明集成 KNN 缺陷预测模型的性能与广泛采用的预测方法相比有明显的提高. 文献 [12] 提出一种分类不平衡影响分析方法,设计了一种新的数据集构造算法,将原不平衡数据集转化为一组不平衡率依次递增的新数据集,实验结果表明代价敏感学习和集成学习在分类不平衡时的性能更优. 文献 [13] 提出一种多分类不平衡的方法,将特征选择与 Boosting 进行结合,实验结果表明,所提出的算法分类精确率更高.

随着半监督方法和集成方法的不断发展,如何将二者结合应用到软件缺陷预测中,成为一个值得探究的内容. 文献 [14] 提出一种基于半监督集成学习的软件缺陷预测方法 SSEL,利用软件中大量存在的未标记样本进行学习,并采用训练样本集权重向量更新策略提升模型的预测性能,实验结果表明,SSEL 模型具有较好的预测效果. 文献 [15] 提出一种基于非负稀疏图的 SemiBoost 软件缺陷预测方法 NSSB,由标记数据、未标记数据、非负稀疏相似度矩阵组成一个提升框架,同时使用集成分类器 Adaboost 提升模型的性能. 实验结果表明,NSSB 方法能有效解决分类不平衡以及标记样本不足的问题. 文献 [16] 提出一种基于搜索的半监督集成跨项目软件缺陷预测方法 S^3EL,实验结果表明,S^3EL 方法在多个公开数据集上与多种典型的跨项目缺陷预测方法相比均能获得较好的预测性能,并且能有效提高朴素贝叶斯的预测能力.

上述方法大部分只针对软件缺陷预测中存在的一个问题进行研究,即只研究标记样本不足或者类分布不平衡的情况,综合考虑两种情况的研究还较少. 而且,在软件缺陷预测中,综合利用 Tri_training 方法、采样方法与集成方法的研究还相对较少. 本文创新性地将半监督 Tri_training 方法与集成 AdaBoost 方法进行结合,同时在预标注样本的过程中加入欠采样方法,并且在对新样本进行预测之前加入 SMOTE 过采样方法. 半监督方法能有效利用未标记样本提升分类器的预测性能,而采样方法与集成方法在解决分类不平衡问题方面能取得较好的效果. 结合这些方法的优势对软件缺陷的倾向性进行分类预测,从而有效提高分类模型的性能.

3　基于机器学习的软件缺陷预测模型

软件缺陷预测是一个二分类问题,其分类结果可以分为有缺陷模块和无缺陷模块,通常使用机器学习方法构建软件缺陷预测模型,如图 1 所示.

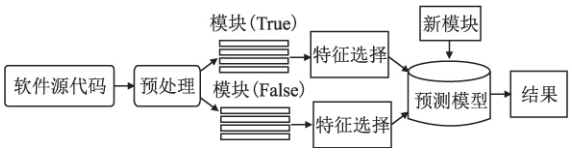

图 1　基于机器学习的软件缺陷预测过程

Fig. 1　Software defect prediction process based on machine learning

图 1 展示了静态软件缺陷预测的一般过程,其中预测模型通常使用机器学习方法构建. 基于机器学习的软件缺陷预测过程如下:

1) 数据获取: 在获取缺陷数据的过程中,首先利用版本控制系统以及缺陷跟踪系统对软件历史仓库中的项目进行属性度量,然后对数据进行归一化、过滤等数据预处理操作.

2) 特征选择: 数据规模过大会对分类结果产生一定的影响,选择合适的特征数据集有助于减少计算量并且能提升分类模型的性能.

3) 缺陷预测: 使用经过调整后的分类模型对每个新的测试模块进行预测,输出结果为 True 或者 False,开发人员可以根据预测结果对缺陷模块进行检查并修改.

4　基于半监督集成学习算法的软件缺陷预测

假设 $L=\{(x_1,y_1),(x_2,y_2)\cdots(x_l,y_l)\}$ 表示有标记样本集,$U=\{x_{l+1},x_{l+2},\cdots,x_{u+l}\}$ 表示无标记样本集. $x\subset X$ 代表软件模块,x 由一组软件度量元 $m=\{d_1,d_2,d_3,\cdots,d_n\}$ ($d_i\in R^d$) 组成. y 代表与模块相对应的缺陷标记,其中 $Y=\{$true, false$\}$,true 表示有缺陷模块,false 表示无缺陷模块. 假设缺陷模块数为 D,无缺陷模块数为 N. 则软件系统中缺陷率和不平衡率定义如下:

定义 1. (缺陷率) 软件系统的缺陷率 D_P 为缺陷模块数与总模块数的比值:

$$D_P=\frac{D}{D+N} \tag{1}$$

定义 2. (不平衡率) 软件系统中的不平衡率 I_P 为无缺陷模块数与有缺陷模块数的比值:

$$I_P=\frac{N}{D} \tag{2}$$

从缺陷率以及不平衡率的定义中能够看出,缺陷模块数越多,缺陷率就越高,数据集的不平衡程度就越小.

4.1　空指针引用缺陷数据集处理

软件缺陷预测结果一般分为有缺陷和无缺陷两类,并没有具体到某一种缺陷类别. 为了更深入的探究软件缺陷预测

模型能否对具体缺陷类别的数据集进行预测,本文使用两种类型的数据集对模型进行验证. 一种是 NASA MDP 数据集,另一种是基于开源项目的包含有空指针引用缺陷的数据集.

空指针引用缺陷(Null Pointer Defect,NPD)是一种常见的代码级别的缺陷. 文献[17]将空指针引用缺陷定义为:引用了可能为 null 的变量. 以 java 语言为例,其产生的主要原因如下:

1)当一个对象不存在时,调用其方法会产生异常,比如 object 是一个不存在的对象,当调用 object. method 时会产生空指针异常.

2)当访问或修改一个对象不存在的字段时会产生异常. 比如 method 对象不存在,则访问 object. method 时会产生空指针异常.

本文初次尝试针对特定的缺陷类别进行探究. 以 java 开源项目为例,使用静态分析工具 SourceMonitor 和缺陷检测工具 Findbugs 对项目中的缺陷进行检测,将检测结果分为:空指针引用缺陷(NPD)模块、无缺陷(No)模块. 经 Findbugs 检测后的开源项目缺陷数量极少,其缺陷率均低于 5%,同时这些缺陷中包含多种不同类别的缺陷. 由于本文只针对空指针缺陷进行研究,因此提出一种将空指针引用缺陷注入开源项目中的方法. 将原缺陷数据集转化为只含有一种空指针缺陷类别的数据集. 其处理过程如图 2 所示.

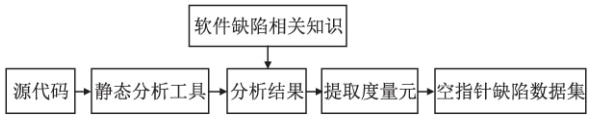

图 2 空指针缺陷数据集生成过程

Fig. 2 Generation process of null pointer defect datasets

为模拟实际项目中不平衡程度较低和较高两种情况,将实验选取的两个开源项目中的空指针引用缺陷率设定为 P,P 取两个不同的值 P_1 和 P_2,P_1 和 P_2 的范围分别设定为:$0 < P_1 < 10\%$,$10\% < P_2 < 20\%$. 引入空指针缺陷的算法如下:

算法 1. 空指针缺陷数据集构造算法

输入: SourceDataSet -开源项目数据集

输出: New SourceDataSet -含有空指针引用缺陷的开源项目数据集

1. 将 SourceDataSet 分为 Defect、NoDefect、NullDefect //将源数据集分为缺陷、无缺陷、空指针缺陷数据集
2. DNum = Defect. size( ) //有缺陷样本数
3. NNum = NoDefect. size( ) //无缺陷样本数
4. NPDNum = NullDefect. size( ) //空指针缺陷样本数
5. OtherNum = DNum - NPDNum //非空指针的其他缺陷样本数
6. DNum = NPDNum + OtherNum //项目中的缺陷数为空指针缺陷数量和非空指针缺陷数量的和
7. $D_p = \frac{DNum}{(DNum + NNum)}$ //开源项目的缺陷率
8. $NPD_p = \frac{NPDNum}{(DNum + NNum)}$ //开源项目中的空指针缺陷率
9. $P = P_i$ $(i = 1, 2)$ //设定注入空指针后总的空指针缺陷率为 P
10. if ($D_p <= P$) //开源项目中的缺陷率小于设定的 P 值
11. if (Defect has noNPD) //缺陷数据集中没有空指针缺陷
12. add D_p and P-D_p NPD into Defect and NoDefect //将一定比例的空指针缺陷注入缺陷和无缺陷数据集中
13. end if
14. else if (Defecthas NPD)
15. add (P-NPD_p) NPD into Defect and NoDefect
16. end if
17. end if
18. if ($D_p > P$) //开源项目中的缺陷率大于设定的 P 值
19. if(Defecthas no NPD)
20. add PNPD into Defect
21. delete (D_p - P) NPD //删除 D_p - P 比例的空指针缺陷
22. end if
23. if(Defect has NullDefect and $P >= NPD_p$)
24. add (P-NPD_p) NPD into Defect
25. end if
26. end if
27. 更新数据集 New SourceDataSet

在算法第 12、15、20、24 步中注入空指针引用缺陷数据集时,选择符合以下规则的模块进行添加:根据开源项目中代码的特征及结构,选择源代码中被赋值为 null 的对象并注入空指针缺陷. 在 Java 中每个类都默认继承 Object 类,除非声明继承某个类. 而 Object 类中有一个叫做 toString 的方法. 该方法返回的是该 Java 对象的内存地址经过哈希算法得出的 int 类型的值再转换成十六进制. 这个输出的结果可以等同的看作 Java 对象在堆中的内存地址. 比如有如下代码,

```
(1) if (arg == null)
      sb. append("nil");
    else
      return 0;
```

则将引入空指针后的代码改为:

```
(2) if (arg == null)
      arg. toString();
      sb. append("nil");
    else
      return 0;
```

以上述方式引入的空指针引用缺陷不会破坏代码的整体结构和功能,同时 Findbug 能够报告出空指针异常,以此使更新后的实验数据集更具有真实性.

4.2 扩充标记样本

Tri-training 算法由 Zhou and Li[18] 提出,是对 1998 年 A. Blum 和 T. Mitchell[19] 提出的协同训练算法 Co-training 的改进. Co-training 算法要求数据集有两个充分冗余的视图,但是在实际应用中很难达到这一要求. Tri-training 算法不要求充分冗余视图、也不要求使用不同类型的分类器,因此其适用范围更加广泛. Tri-training 算法使用投票法将三个分类器进行组合之后对未标记样本进行预测,投票法能有效减少不相关特征和冗余特征对预测模型性能的影响.

Tri-training 算法扩充标记样本时,首先由其中任意两个分类器 h_1 和 h_2 对任意一个无标记样本 $x \in U$ 进行标记. 如果两个分类器对 x 的标记结果相同,则将其中一个标记结果作为分类器 h_3 的训练样本,即:$h_1(x) = h_2(x)$,则将 $L_i = \{(x, h_2(x))\}$,$x \in U$ 加入 h_3 的训练集中.

h_3在训练的过程中若能达到较高的准确度,则训练结果会得到优化; 反之,会将噪声引入 h_3的训练集中,降低分类器的标记性能. 基于此,Zhou 等人[18] 作出如下证明:

$$|L\cup L^t|\left(1-2\frac{\eta_L|L|+\tilde{e}_1^t|L^t|}{|L\cup L^t|}\right)^2>|L\cup L^t|\left(1-2\frac{\eta_L|L|+\tilde{e}_1^{t-1}|L^{t-1}|}{|L\cup L^{t-1}|}\right)^2 \quad (3)$$

在 PAC 可学习框架下,如果新标记的训练样本足够多并且能够满足公式(3) 中的条件,那么 h_3再次进行训练时所得假设的分类性能会提高. 其中,

L^t: 第 t 次迭代时 h_1、h_2为 h_3新标记的训练样本

$\tilde{e}_1{}^t$: 第 t 次迭代时 h_1、h_2标记的样本中被错误标记数量所占的比值

η_L: 初始训练集 L 的噪声率

第 t 轮迭代后的分类噪声率可由如下公式表示:

$$\eta^t=\frac{\eta|L|+\tilde{e}_1{}^t|L^t|}{|L\cup L^t|} \quad (4)$$

当 $0<\tilde{e}_1{}^t,\tilde{e}_1{}^{t-1}<0.5$,且 $|L^t|>|L^{t-1}|$时,可得出:

$$\tilde{e}_1{}^t|L^t|<\tilde{e}_1{}^{t-1}|L^{t-1}| \quad (5)$$

因此,公式(3) 可变换如下:

$$0<\frac{\tilde{e}_1{}^t}{\tilde{e}_1{}^{t-1}}<\frac{|L^t-1|}{L^t} \quad (6)$$

对标记样本进行扩充的过程中,可用公式(6) 判断经分类器 h_1和 h_2标记的样本$\{x,h_2(x)\}$能否加入分类器 h_3中作为新一轮的训练样本.

由于 Tri-training 对样本进行预标注时需要使用少量的标记样本和大量的未标记样本,但是标记样本不仅数量少,而且也是分类不平衡的. 为了提高预标注样本的准确率,在 Tri-training 算法中加入欠采样方法. 通过随机删除训练集中的无缺陷样本提升预标注样本的准确率. 欠采样方法的伪代码如下:

算法 2.　UnderSampling 算法

输入: 训练数据集 S

输出: 采样后的训练集 S_deal

```
1. 正类样本集 Y,负类样本集 N,S = Y + N
2. for x_i ∈ N do
3.    for x_j ∈ N do
4.       for x_k ∈ S do
5.          Find = false
6.          if d(x_i, x_k) <d(x_i, x_j) or d(x_j, x_k) <d(x_i, x_j) //样本点之间的距离
7.             Find = true
8.             break
9.          end if
10.       end for
11.       if (Find == true)
12.          delete x_j //删除多数类样本
13. end if
14.    end for
15. end for
16. 更新数据集 S,S_deal = S
```

4.3　Tri_AdaBoost 分类模型

SMOTE(Synthetic Minority OverSampling) 算法由 Chawla 在 2002 年提出[20]. SMOTE 是基于随机过采样算法的一种改进方案,其基本思想是对少数类样本进行分析并根据少数类样本人工合成新样本添加到数据集中. 由于经过扩充后的样本依然是一个不平衡的数据集,因此,为了提高预测模型的性能,在对新数据集进行分类预测之前,使用 SMOTE 算法对其进行采样.

AdaBoost(Adaptive Boost) 算法由 Freund 和 Schapire 在 1996 年提出[21]. AdaBoost 是一种集成学习算法,集成学习就是通过综合多个分类器的预测结果来提高分类准确率. 该算法的核心思想是针对同一个训练集训练不同的弱分类器,它根据每次训练集中每个样本的分类是否正确,以及样本的分类误差,来确定下一次迭代时每个样本的权值. 最后将每次训练得到的分类器按一定的权重融合到一起,形成强分类器并将其作为最终的决策分类器.

AdaBoost 算法使用加法模型,如公式(7) 所示. 损失函数为指数函数,如公式(8) 所示. 学习算法使用前向分步算法,强学习器通过前向分步算法得到.

加法模型:

$$f(x)=\sum_{m=1}^{M}\alpha_m G_m(x) \quad (7)$$

损失函数:

$$Ly,f(x)=\exp(-yf(x)) \quad (8)$$

AdaBoost 能够提高分类的准确性可能有如下两个原因:

1) 通过将具有较高误分类率的分类器进行组合,减少了最终分类器的误分类率. Freud 在文献[22] 中证明: 假设 AdaBoost 每一轮迭代中生成的子分类器错误率分别为: $\epsilon_1,\epsilon_2,\cdots,\epsilon_R$,则集成分类器 H 的训练错误率有上界:

$$\epsilon\leqslant 2^R\prod_r^R\sqrt{\epsilon_r(1-\epsilon_r)} \quad (9)$$

2) 最终组合分类器的方差小于弱分类器的方差.

结合半监督算法 Tri_training 与集成算法 Adaboost 的优势,对 AdaBoost 算法进行改进,提出一种新的分类算法 Tri_AdaBoost(Tri_training and Adaboost). 该算法随机选取一部分未标记数据,利用 Tri-Training 算法以及欠采样方法对未标记数据集进行预标注,将预标注的数据集扩充到原始的已标记数据集中形成新的标记样本; 再利用 SMOTE 算法对新的样本集进行采样,然后使用采样后的数据集训练 AdaBoost 分类器; 最后,对测试集进行预测并分类. Tri_AdaBoost 预测模型如图 3 所示.

4.4　Tri_AdaBoost 分类算法

在集成算法 AdaBoost 中,随着分类器的增加,损失函数会逐渐减小,即偏差会越来越小,方差逐渐增大. 因此与构成 AdaBoost 的基础分类模型相比,AdaBoost 更容易过拟合,但是当构成 AdaBoost 的基分类模型比较简单时,AdaBoost 模型很难发生过拟合,因此本文将单层决策树 DecisionStump 作为基础分类器. 使用 SMOTE 方法对新数据集进行采样时,过采样倍率 N 设置为 1,k 值取 5.

Tri_AdaBoost 分类算法步骤如算法 3 所示.

算法 3.　Tri_AdaBoost 算法

输入: 有标记训练集 $L=\{(x_1,y_1),(x_2,y_2)\ldots(x_l,y_l)\}$,无标记训练集 $U=\{(x_{l+1}),(x_{l+2})\ldots(x_{l+u})\}$,监督学习算法

Learn

1. S_deal = L + U

2. UnderSampling (L) →L′

3. Bootstrap(L′) → L1′,L2′,L3′

4. use Learn and L1′,L2′,L3′ train classifier h_1 ,h_2 ,h_3

5. $e_1′ = e_2′ = e_3′ = 0.5$,$l_1′ = l_2′ = l_3′ = 0$

6. repeat training h_1 ,h_2 ,h_3

7. h_i label software modules in U,as $L′_t$

8. for $i \in \{1 \cdots 3\}$ do

9. $e_i \leftarrow$MeasureError(h_j & h_k) ($j,k \neq i$)

10. if $e_i < e′_i$ then

11. use h_j and h_k to Determine and mark software modules in U,add them into $l′_i$

12. end if

13. until h_j no longer changes

14. return $l′_i$

15. end for

16. n = L′ + $l′_i$ //更新有标记训练集 L 的样本数量

17. SMOTE(D,N,k) //D 为正类样例数目,N 为采样倍率,k 为最近邻数

18. L = { $L_j = 1/n′ \mid j = 1,2,\cdots,n′$} //设置标记样本 L 中每个样本的权值,经 SMOTE 采样后的标记样本数量设为 $n′$

19. for i = 1 to T do // T 表示迭代次数

20. train weak classifier h_t

21. $\alpha_i = \frac{1}{2}\ln\frac{1-e_i}{e_i}$ //计算弱分类器的权重

22. $w_j^{(i+1)} = \frac{w_j^{(i)}}{z_j} \times \begin{cases} e^{-a_i} & \text{如果 } c_i(x_i) = y_i \\ e^{a_i} & \text{如果 } c_i(x_i) \neq y_i \end{cases}$ //更新各样本的权值

23. end for

24. return $Y_M(x) = \text{sign}(\sum_{m=1}^{M} \alpha_m G_m(x))$

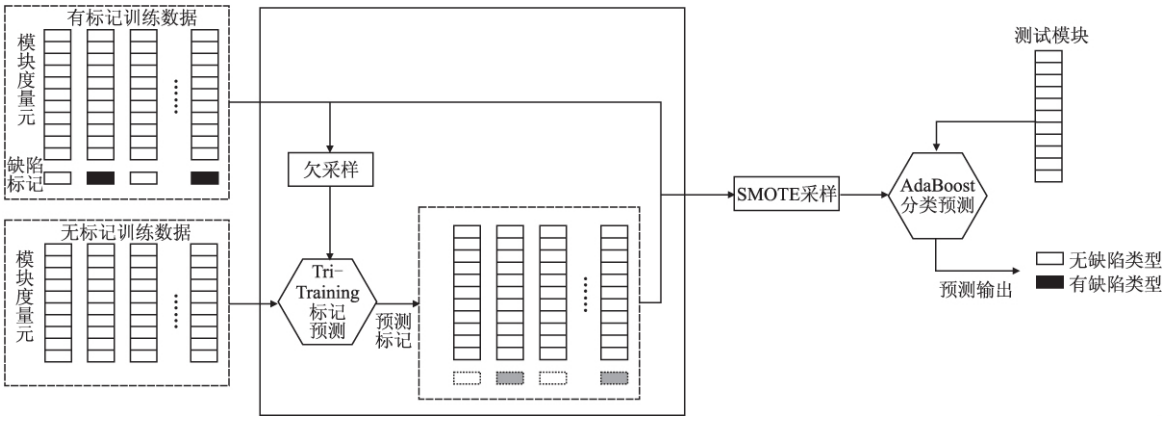

图 3 Tri_AdaBoost 分类模型的过程

Fig. 3 Process of Tri_AdaBoost classification model

5 实 验

5.1 数据集

实验数据集由两部分组成.

1) 选取开放软件缺陷数据库 PROMISE 库[23] 中 4 个项目的缺陷数据集进行实验,分别是: PC1,PC3,PC4,KC2,这些数据集均由美国航空航天局(NASA) 提供. 项目分别采用C 语言或 java 语言实现,其中包含了对数据集中各模块有缺陷或者无缺陷标注情况的描述以及对数据集的属性说明,数据集中的属性由 McCabe、Halstead 等度量方法生成,McCabe、Halstead 能客观的表征与软件质量相关联的质量特征. 表 1 给出了 4 个项目的基本信息,包括项目名称、度量元个数、模块总数、缺陷模块数以及缺陷率.

表 1 NASA MDP 数据集基本信息

Table 1 Basic information of NASA MDP datasets

项目	度量元数	模块总数	缺陷模块数	缺陷率
PC1	38	705	61	8.65%
PC3	38	1077	134	12.44%
PC4	38	1458	178	12.21%
KC2	22	522	107	20.50%

2) 使用 Findbugs 缺陷检测工具对 aTunes、Nutch 两个开源项目进行检测,经过分析总结之后其缺陷数据集信息如表 2 所示,经过算法 1 更新后的实验数据集如表 3 所示.

表 2 开源项目数据集基本信息

Table 2 Basic information of open source project datasets

项目	度量元数	文件数	缺陷总数	缺陷率
aTunes	19	216	10	4.60%
Nutch	20	576	13	2.00%

表 3 更新后的开源项目数据集基本信息

Table 3 Basic information updated open source project datasets

项目	度量元数	文件数	空指针缺陷数	缺陷率
aTunes	19	216	26	12.03%
Nutch	20	576	24	4.17%

表 4 混淆矩阵

Table 4 Confusion matrix

	预测为正类	预测为负类
实际为正类	TP	FN
实际为负类	FP	TN

5.2 评价指标

软件缺陷预测的结果可以用含混矩阵表示. 如表 4 所示,

以 TP、TN、FP、FN 表示分类结果. TP 表示被正确分类的正类样本,TN 表示被正确分类的负类样本,FP 表示实际为负类的样本被错误分类为正类样本,FN 表示实际为正类的样本被错误分类为负类样本. 预测结果为正类则表示模块是有缺陷的,预测结果为负类则表示模块是无缺陷的.

为了有效评价分类器的性能及分类不平衡对分类结果的影响,采用召回率、精确度、F-measure 值、准确率和 AUC 值 5 个评价指标对实验结果进行分析. AUC 作为数值可以直观的评价分类器的性能,是用来衡量缺陷预测模型好坏的一个标准. 各评价指标可表示如下:

1) 召回率(Recall): 被模型正确预测的正类样本占实际总的正类样本的比例

$$Recall = \frac{TP}{(TP + FN)} \tag{10}$$

2) AUC: 受试者工作特征(Receiver Operating Characteristic,ROC) 曲线下方的面积. ROC 曲线的横坐标为 FPR (false positive rate),即实际为负类的样本被错误分类为正类的样本占实际总的负类样本的比例. 纵坐标为 TPR (true positive rate),即被正确分类的正类样本占实际总的正类样本的比例. ROC 曲线越接近左上角,代表结果越好. 即 AUC 值越大,分类器性能越好.

3) 准确率(Accuracy): 被模型正确分类的样本(正类和负类) 占实际总样本数的比例

$$Accuracy = \frac{TP + TN}{TP + FN + TN + FN)} \tag{11}$$

4) 精度(Precision): 被模型正确分类的正类样本占被正确分类的正类样本与被错误分类的负类样本和的比例

$$Precision = \frac{TP}{(TP + FP)} \tag{12}$$

5) F-measure: 召回率和精度有时候会出现矛盾的情况,这样就需要综合考虑二者. 最常见的方法是 F-measure,它是召回率和精度的加权调和平均值. F-measure 值较高时,说明模型性能较好.

$$F = \frac{2 \times Recall \times Precision}{(Recall + Precision)} \tag{13}$$

5.3　数据集实验结果及分析

本节分别对 NASA MDP 数据集和基于开源项目生成的空指针缺陷数据集进行分析. 选取 J48、NaiveBayes 两种经典机器学习方法[24] 以及 Adaboost、RandomForest 两种集成学习方法与 Tri_Adaboost 算法进行对比,实验所用对比方法的运行参数均取 weka 中的默认值.

本文实验设置如下,将每一个数据集按照 3∶2 的比例划分成训练集和测试集,再根据特定的标记率 L 将训练集划分成有标记样本和无标记样本,本文将 L 的标记率分别设定为 0.2、0.3、0.4 三种. 对于本文的方法以及选用的对比方法采取 10 折交叉验证的方式重复进行 10 次实验. 比如,对于 Tri_Adaboost 方法,假设数据集中有 100 个数据样本,从这些样本中随机选取 60 个样本作为半监督阶段的训练集,对其余的 40 个样本进行标记,最后对所有带标记的样本进行 10 折交叉验证,选取其中 9 份作为训练集,1 份作为测试集.

5.3.1　NASA MDP 数据集实验结果及分析

如表 5 所示为各分类模型在 NASA MDP 数据集上的实验结果.

从表 5 中可以看出,在 3 种标记率下进行的共 60 次评估结果中,半监督集成学习算法(Tri_Adaboost) 共获得 52 次最优值,占比 86.7%. 其中,

1) 在 PC1 数据集上,与其他四种方法相比,Tri_Adaboost 方法在三种标记率下均获得最优值.

2) 在 PC3 数据集上,虽然 Tri_Adaboost 方法不能一直获得最优值,但是其 Accuracy 和 Recall 值与其他几种方法的差值最大分别为 0.033(0.878-0.845)、0.126(0.690-0.564),依然能得到与对比方法可比的结果. 由于三种标记率下,Tri_Adaboost 在 AUC 以及 F-measure 上均获得最优值. 因此,总体来说,Tri_Adaboost 的预测性能优于其他四种方法.

3) 在 PC4 数据集上,Tri_Adaboost 方法在 Accuracy 和 Precision 上不能一直获得最优值,但是其 AUC 以及 F-measure 值均高于对比方法. 因此,Tri_Adaboost 的预测性能依然优于其他四种方法.

4) 在 KC2 数据集上,Tri_Adaboost 方法在三种标记率下均获得最优值. 从平均值能够看出,Tri_Adaboost 方法在各个评价指标上的均值都高于其他几种方法.

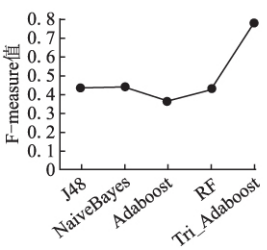

图 4　各方法在三种标记率下的 F-measure 平均值

Fig. 4　F-measure mean of each method under three mark rates

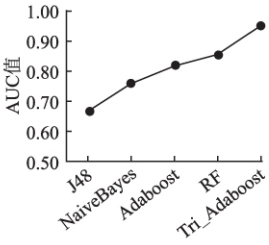

图 5　各方法在三种标记率下的 AUC 平均值

Fig. 5　AUC mean of each method under three mark rates

为了更直观的看出各分类模型的分类效果,以折线图表示各分类模型在几种标记率下的 F-measure 和 AUC 平均值. 从图 4 和图 5 中可以看出,与其他几种方法相比,Tri_Adaboost 方法的 F-measure 和 AUC 最优. 同时,其 F-measure 和 AUC 值分别至少提高了 0.344(0.779-0.435)、0.099(0.952-0.853). 结合表 5 中的实验结果可以得出,标记样本的扩充以及未标记样本的使用在一定程度上能提升预测模型的性能,Tri_Adaboost 方法是一种有效的半监督集成学习方法,并且具备比其他几种方法较强的学习能力.

5.3.2　基于开源项目下的空指针引用缺陷数据集实验结果及分析

如表 6 所示为各分类方法在引入空指针缺陷数据集后的实验结果.

从表 6 中可以看出,Tri_Adaboost 在大部分情况下的评价指标值都高于其他方法. aTunes 和 Nutch 在三种标记率下的 AUC 值都高于其他方法,即使在 Nutch 的不平衡率较高的情况下,本文的方法依然能获得较高的 AUC 值,说明 Tri_Adaboost 方法能有效的解决分类不平衡的问题. 同时,aTunes 和 Nutch 在三种标记率下的 F-measure 值均优于对比方法,说明本文所提出的模型性能较好.

表 5 不同样本标记率下各分类方法在 NASA MDP 数据集上的实验结果

Table 5 Experimental results of various classification methods on NASA MDP datasets under different sample labeling rates

数据集	评价指标	Accuracy			Precision			Recall			F-Measure			AUC		
		L=0.2	L=0.3	L=0.4	L=0.2	L=0.3	L=0.4	L=0.2	L=0.3	L=0.4	L=0.2	L=0.3	L=0.4	L=0.2	L=0.3	L=0.4
PC1	J48	0.891	0.882	0.897	0.385	0.174	0.450	0.270	0.108	0.188	0.317	0.133	0.265	0.640	0.495	0.605
	NaiveBayes	0.891	0.900	0.872	0.429	0.415	0.354	0.486	0.459	0.354	0.456	0.436	0.354	0.781	0.784	0.732
	Adaboost	0.899	0.895	0.899	0.435	0.286	0.444	0.270	0.162	0.083	0.333	0.207	0.140	0.781	0.740	0.781
	RF	0.906	0.914	0.899	0.500	0.462	0.474	0.162	0.162	0.188	0.245	0.240	0.269	0.812	0.857	0.819
	Tri_Adaboost	0.956	0.963	0.919	0.700	0.862	0.720	0.538	0.658	0.563	0.609	0.746	0.632	0.950	0.963	0.921
PC3	J48	0.845	0.867	0.852	0.339	0.446	0.405	0.313	0.382	0.368	0.326	0.411	0.386	0.577	0.664	0.702
	NaiveBayes	0.775	0.733	0.740	0.285	0.264	0.283	0.582	0.671	0.690	0.382	0.379	0.401	0.770	0.768	0.776
	Adaboost	0.880	0.879	0.875	0.500	0.000	0.571	0.060	0.000	0.046	0.107	0.000	0.085	0.811	0.819	0.802
	RF	0.873	0.888	0.878	0.389	0.607	0.552	0.104	0.224	0.184	0.165	0.327	0.276	0.816	0.862	0.835
	Tri_Adaboost	0.871	0.861	0.845	0.729	0.721	0.683	0.709	0.631	0.564	0.719	0.673	0.618	0.927	0.913	0.877
PC4	J48	0.882	0.899	0.883	0.511	0.602	0.523	0.516	0.585	0.509	0.514	0.593	0.516	0.736	0.759	0.740
	NaiveBayes	0.852	0.865	0.869	0.402	0.451	0.450	0.473	0.349	0.316	0.434	0.394	0.371	0.836	0.833	0.833
	Adaboost	0.897	0.895	0.909	0.627	0.622	0.693	0.352	0.434	0.456	0.451	0.511	0.550	0.911	0.912	0.922
	RF	0.902	0.916	0.902	0.667	0.787	0.662	0.374	0.453	0.412	0.479	0.575	0.508	0.933	0.948	0.940
	Tri_Adaboost	0.973	0.911	0.899	0.929	0.854	0.679	0.858	0.761	0.824	0.892	0.805	0.744	0.982	0.961	0.949
KC2	J48	0.734	0.690	0.743	0.692	0.573	0.624	0.589	0.477	0.495	0.636	0.520	0.552	0.690	0.652	0.684
	NaiveBayes	0.734	0.739	0.763	0.818	0.741	0.726	0.421	0.402	0.421	0.556	0.521	0.533	0.788	0.800	0.812
	Adaboost	0.720	0.746	0.746	0.632	0.625	0.590	0.692	0.701	0.673	0.661	0.661	0.629	0.747	0.787	0.763
	RF	0.749	0.749	0.784	0.701	0.663	0.692	0.636	0.589	0.589	0.667	0.624	0.636	0.803	0.799	0.812
	Tri_Adaboost	0.987	0.991	0.989	0.989	0.984	0.976	0.969	0.969	0.932	0.979	0.976	0.953	0.995	0.996	0.996
平均值	J48		0.840			0.477			0.400			0.431			0.662	
	NaiveBayes		0.811			0.468			0.469			0.435			0.755	
	Adaboost		0.853			0.502			0.327			0.361			0.815	
	RF		0.863			0.596			0.340			0.418			0.853	
	Tri_Adaboost		**0.930**			**0.819**			**0.748**			**0.779**			**0.952**	

表 6 不同样本标记率下各分类方法在开源项目数据集上的实验结果

Table 6 Experimental results of various classification methods on data sets of open source projects under different sample mark rates

数据集	评价指标	Accuracy			Precision			Recall			F-Measure			AUC		
		L=0.2	L=0.3	L=0.4	L=0.2	L=0.3	L=0.4	L=0.2	L=0.3	L=0.4	L=0.2	L=0.3	L=0.4	L=0.2	L=0.3	L=0.4
aTunes	J48	0.853	0.865	0.831	0.500	0.533	0.450	0.467	0.500	0.500	0.483	0.516	0.474	0.656	0.701	0.752
	NaiveBayes	0.833	0.820	0.814	0.450	0.417	0.417	0.600	0.625	0.556	0.514	0.500	0.476	0.814	0.823	0.836
	Adaboost	0.882	0.892	0.873	0.588	0.643	0.600	0.667	0.563	0.500	0.625	0.600	0.545	0.811	0.838	0.843
	RF	0.912	0.937	0.949	0.800	0.846	0.875	0.533	0.688	0.778	0.640	0.759	0.824	0.863	0.859	0.883
	Tri_Adaboost	0.962	0.984	0.926	0.947	0.972	0.930	0.964	0.972	0.883	0.956	0.972	0.906	0.982	0.990	0.981
Nutch	J48	0.953	0.952	0.946	0.667	0.750	0.474	0.444	0.167	0.474	0.533	0.273	0.474	0.838	0.760	0.798
	NaiveBayes	0.732	0.637	0.631	0.131	0.094	0.091	0.611	0.667	0.684	0.216	0.166	0.160	0.740	0.754	0.740
	Adaboost	0.933	0.946	0.943	0.400	0.500	0.250	0.222	0.111	0.053	0.286	0.182	0.087	0.845	0.822	0.863
	RF	0.946	0.952	0.951	1.000	1.000	0.667	0.111	0.111	0.105	0.200	0.200	0.182	0.887	0.904	0.860
	Tri_Adaboost	0.957	0.980	0.956	0.939	0.973	0.774	0.912	0.857	0.706	0.925	0.911	0.738	0.996	0.991	0.958
平均值	J48		0.900			0.562			0.425			0.459			0.751	
	NaiveBayes		0.744			0.267			0.624			0.339			0.784	
	Adaboost		0.912			0.497			0.353			0.388			0.837	
	RF		0.941			0.865			0.388			0.484			0.876	
	Tri_Adaboost		**0.961**			**0.922**			**0.882**			**0.901**			**0.983**	

以折线图表示各分类方法在两个空指针缺陷数据集上的 F-meausre 和 AUC 平均值. 从图中可以看出,与其他几种方法相比,Tri_Adaboost 方法在两个评价指标上的平均值能获得最高值. 因此,在空指针引用缺陷数据上的实验结果依然能表明本文方法的有效性.

6 结束语

针对软件缺陷预测中标记样本难以获取以及类别分布不平衡的情况,本文提出了一种基于半监督 Adaboost 集成方法的软件缺陷预测模型,将 Tri_training 与 AdaBoost 两种算法进

行结合形成一种新的分类框架. 同时在扩充标记样本的过程中加入欠采样方法,以降低Tri_training标记样本时的错误率;在分类预测之前,对扩充样本集进行 SMOTE 采样,以降低新样本集的不平衡率,提升模型的性能. 实验中除使用 NASA MDP 数据集进行验证外,首次尝试使用基于开源项目生成的空指针引用缺陷数据集进行了实验.

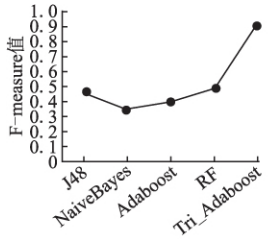

图 6　各方法在三种标记率下的 F-measure 平均值

Fig. 6　F-measure mean of each method under three mark rates

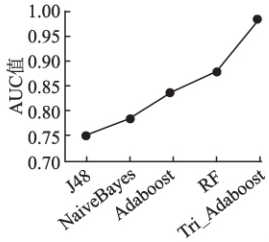

图 7　各方法在三种标记率下的 AUC 平均值

Fig. 7　AUC mean of each method under three mark rates

从实验结果可以看出,本文提出的Tri_AdaBoost 方法在不同比例的标记样本下的预测效果均优于 J48、NaiveBayes、Adaboost 以及 RF. 下一步工作的重点将探究度量元对软件缺陷预测的影响,并且将尝试加入其它类别的软件缺陷进行实验,同时会进一步探究无监督方法与集成方法或者半监督方法与 Bagging 等集成方法的结合对软件缺陷预测模型的影响.

References:

[1] Huizinga D, Kolawa A. Automateddefect prevention : best practices in software management [M]. IEEE Xplore, 2007.

[2] Chen Xiang, Gu Qing, Liu Wang-shu, et al. Survey of static software defect prediction [J]. Journal of Software, 2016, 27(1): 1-25.

[3] Menzies T, Dekhtyar A, Distefano J, et al. Problems with precision: a response to "comments on 'data mining static code attributes to learn defect predictors'" [J]. IEEE Transactions on Software Engineering, 2007, 33(9): 1-4.

[4] Galar M, Fernandez A, Barrenechea E, et al. A review on ensembles for the class imbalance problem: bagging-, boosting-, and hybrid-based approaches [J]. IEEE Transactions on Systems Man & Cybernetics Part C Applications & Reviews, 2012, 42(4): 463-484.

[5] Siers M J, Islam M Z. Software defect prediction using a cost sensitive decision forest and voting, and a potential solution to the class imbalance problem [J]. Information Systems, 2015, 51(C): 62-71.

[6] Tahir M A, Kittler J, Yan F. Inverse random under sampling for class imbalance problem and its application to multi-label classification [J]. Pattern Recognition, 2012, 45(10): 3738-3750.

[7] Ma Y, Pan W, Zhu S, et al. An improved semi-supervised learning method for software defect prediction [J]. Journal of Intelligent & Fuzzy Systems, 2014, 27(5): 2473-2480.

[8] Catal C, Diri B. Unlabelled extra data do not always mean extra performance for semi-supervised fault prediction [J]. Expert Systems, 2009, 26(5): 458-471.

[9] Lu H, Cukic B, Culp M. An iterative semi-supervised approach to software fault prediction [C]. International Conference on Predictive MODELS in Software Engineering, Promise 2011, Banff, Alberta, Canada, September, DBLP, 2011.

[10] Seliya N, Khoshgoftaar T M. Software quality estimation with limited fault data: a semi-supervised learning perspective [J]. Software Quality Journal, 2007, 15(3): 327-344.

[11] He Liang, Song Qin-bao, Shen Jun-yi. Boosting-based k-NN learning for software defect prediction [J]. Pattern Recognition and Artificial Intelligence, 2012, 25(5): 792-802.

[12] Yu Qiao, Jiang Shu-juan, Zhang Yan-mei, et al. The impact study of class imbalance on the performance of software defect prediction models [J]. Chinese Journal of Computers, 2016, 39(108): 1-18.

[13] Haixiang G, Yijing L, Yanan L, et al. BPSO-adaboost-KNN ensemble learning algorithm for multi-class imbalanced data classification [J]. Engineering Applications of Artificial Intelligence, 2016, 49(C): 176-193.

[14] Wang Tie-jian, Wu Fei, Jing Xiao-yuan. Semi-supervised ensemble learning based software defect prediction [J]. Pattern Recognition and Artificial Intelligence, 2017, 30(7): 646-652.

[15] Wang T, Zhang Z, Jing X, et al. Non-negative sparse-based SemiBoost for software defect prediction [J]. Software Testing Verification & Reliability, 2016, 26(7): 498-515.

[16] He Ji-yuan, Meng Zhao-peng, Chen Xiang, et al. Semi-supervised ensemble learning approach for cross-project defect prediction [J]. Journal of Software, 2017, 28(6): 1455-1473.

[17] Wang Ya-wen. Research on software testing technology based on defect pattern [D]. Beijing: Beijing University of Posts and Telecommunications, 2009.

[18] Zhou Z H, Li M. Tri-training: exploiting unlabeled data using three classifiers [J]. IEEE Trans on Knowledge and Data Engineering, 2005, 17(11): 1529-1541.

[19] Blum A, Mitchell T. Combining labeled and unlabeled data with co-training [C]. Eleventh Conference on Computational Learning Theory, ACM, 2000: 92-100.

[20] 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(1): 321-357.

[21] Freund Y. Experiments with a new boosting algorithm [J]. Proceedings of Icml, 1996, 13: 148-156.

[22] Freund, Yoav, Schapire, Robert E. Adecision-theoretic generalization of on-line learning and an application to boosting [J]. Journal of Computer & System Sciences, 1997, 55(1): 119-139.

[23] Menzies T, Krishna R, Pryor D. The promise repository of empirical software engineering data [EB/OL], http://openscience.us/repo. North Carolina State University, Department of Computer Science, 2016.

[24] Menzies T, Greenwald J, Frank A. Data mining static code attributes to learn defect predictors [J]. IEEE Transactions on Software Engineering, 2006, 33(1): 2-13.

附中文参考文献:

[2] 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.

[11] 何　亮,宋擒豹,沈钧毅. 基于 Boosting 的集成 k-NN 软件缺陷预测方法 [J]. 模式识别与人工智能, 2012, 25(5): 792-802.

[12] 于　巧,姜淑娟,张艳梅,等. 分类不平衡对软件缺陷预测模型性能的影响研究 [J]. 计算机学报, 2016, 39(108): 1-18.

[14] 王铁建,吴　飞,荆晓远. 基于半监督集成学习的软件缺陷预测 [J]. 模式识别与人工智能, 2017, 30(7): 646-652.

[16] 何吉元,孟昭鹏,陈　翔,等. 一种半监督集成跨项目软件缺陷预测方法 [J]. 软件学报, 2017, 28(6): 1455-1473.

[17] 王雅文. 基于缺陷模式的软件测试技术研究 [D]. 北京: 北京邮电大学, 2009.