DOI:10.16451/j.cnki.issn1003-6059.2012.05.006



# 基于 Boosting 的集成 $k$-NN 软件缺陷预测方法*

何 亮 宋擒豹 沈钧毅

(西安交通大学 电子与信息工程学院 西安 710049)

**摘 要** 软件缺陷预测是改善软件开发质量,提高测试效率的重要途径. 文中提出一种基于软件度量元的集成$k$-NN 软件缺陷预测方法. 首先,该方法在不同的 Bootstrap 抽样数据集上迭代训练生成一个基本 $k$-NN 预测器集合. 然后,这些基本预测器分别对软件模块进行独立预测,各基本预测值将被融合生成最终的预测结果. 为判别新的软件模块是否为缺陷模块,设计分类阈值的自适应学习方法. 集成预测结果大于该阈值的模块将被识别为缺陷模块,反之则为正常模块. NASA MDP 及 PROMISE AR 标准软件缺陷数据集上的实验结果表明集成 $k$-NN 缺陷预测的性能较之广泛采用的对比缺陷预测方法有较明显的提高,同时也证明软件度量元在缺陷预测中的有效性.

**关键词** 软件缺陷预测,$k$-近邻($k$-NN),软件度量元,集成学习

**中图法分类号** TP 311.5

# Boosting-Based $k$-NN Learning for Software Defect Prediction

HE Liang, SONG Qin-Bao, SHEN Jun-Yi

(*School of Electronic and Information Engineering, Xi'an Jiaotong University, Xi'an* 710049)

## ABSTRACT

Timely identification of defective modules improves both software quality and testing efficiency. A software metrics-based ensemble $k$-NN algorithm is proposed for software defect prediction. Firstly, a set of base $k$-NN predictors is constructed iteratively from different bootstrap sampling datasets. Next, the base $k$-NN predictors estimate the software module independently and their individual outputs are combined as the composite result. Then, an adaptive threshold training approach is designed for the ensemble to classify new software modules. If the composite result is greater than the threshold value, the software module is recognized as defective, otherwise as normal. Finally, the experiments are conducted on NASA MDP and PROMISE AR datasets. Compared with a widely referenced defect prediction approach, the results show the considerable improvements of the ensemble $k$-NN and prove the effectiveness of software metrics in defect prediction.

**Key Words** Software Defect Prediction, $k$ Nearest Neighbor ($k$-NN), Software Metric, Ensemble Learning

* 国家自然科学基金资助项目(No. 90718024)
收稿日期:2011-05-18;修回日期:2012-02-23
作者简介 何亮,男,1975 年生,博士研究生,主要研究方向为数据挖掘、软件工程. E-mail: lhe@mail.xjtu.edu.cn. 宋擒豹,男,1966 年生,教授,博士生导师,主要研究方向为数据挖掘、机器学习、可信计算等. 沈钧毅,男,1941 年生,教授,博士生导师,主要研究方向为数据库、数据挖掘.

## 1　引　言

软件在应用过程中不可避免的会出现系统故障甚至失效的情况. 这些问题可能是由软件的硬件环境引起,也可能是人为因素造成,而软件系统中隐含的内在错误,即软件缺陷也是导致这一问题的重要因素之一. 以人工开发方式设计实现的软件系统中存在着或多或少的错误,这些错误是导致应用系统故障、失效、崩溃、甚至出现灾难性后果的重要根源. 软件缺陷成为软件系统开发过程中客观存在的一个组成部分,如何尽可能准确地检测出软件系统中的潜在缺陷并加以及时修正是软件工程领域的热点研究课题. 软件缺陷预测技术在这一过程中发挥重要作用,项目开发人员可借助缺陷预测的结果对测试资源进行合理分配,这样不仅有助于检测出更多的潜在缺陷,同时也有利于软件开发成本的降低.

影响软件质量的因素涉及众多方面,开发人员能力水平、采用的开发工具环境、项目类别与难易程度等差异均会对缺陷的产生造成一定影响. 早期的缺陷预测多通过经验分析给出一个笼统的经验估算公式,这样的预测方法往往有一定的局限性,其准确性也不理想. 随着各种机器学习、数据挖掘技术的出现,这些理论及方法逐渐被广泛应用于软件缺陷预测中,缺陷预测的方式也从早期的经验公式估算发展为对缺陷分布的预测.

软件由若干行程序代码编制而成,软件代码的规模、复杂度等度量元(Software Metrics)数据是进行软件缺陷预测的重要依据. 基于度量元的软件缺陷预测是该研究领域中普遍采用的方法,该方法通过对历史项目度量元数据的积累和学习,对新开发系统中可能含有缺陷的软件模块分布进行预测. 尽管有学者对度量元在软件缺陷预测中的作用提出过质疑,不过已有大量研究证明其有效性[1-4].

目前,不同类型的分类技术已被尝试应用于软件缺陷预测,如基于案例的推理方法[5-6]、朴素贝叶斯方法[7]、神经网络方法[8-9]、决策树方法[10-12]、支持向量机[13]等,这些方法以二元分类的形式对软件模块是否含有潜在缺陷进行判别. 通常一个软件系统中的大部分模块均为无缺陷的正常模块,缺陷仅存在于小部分模块中,然而这些缺陷却会对系统的可靠性产生重要影响. 因此,为了对缺陷模块予以准确识别,预测时所采用的模型与方法应能较好地适用于不平衡类问题的处理. 在实际应用中,不同分类技术对缺陷模块的倾斜分布适应能力并不相同. 例如,文献[13]的实验表明支持向量机方法在缺陷预测时取得较好的评估结果. 然而,其评价方式并非针对缺陷模块的正确识别,而是面向大量存在的无缺陷模块的识别. 显然,在缺陷预测中前者比后者有更重要的实际意义,当以前者作为评估标准时,甚至可能会得出相反的预测性能评价结果,本文的后续实验也验证这一点.

基于软件度量元的预测方法已在软件工程领域获得众多关注,例如软件开发工作量的预测及软件缺陷的预测等. Shepperd 和 Schofield[14] 将基于实例的学习方法成功应用于软件项目工作量预测,该研究中各软件项目即以软件特征及度量元描述. $k$ 近邻($k$ Nearest Neighbor, $k$-NN)算法是一种基于实例的消极学习方法,以 $k$ 个近邻的投票或加权和实现未知实例的分类或数值预测. 本文将基于 Boosting 的集成 $k$-NN 方法引入软件缺陷预测应用,通过已开发软件模块度量元数据的学习对新的模块缺陷分布做出预测. 现有的缺陷预测方法一般按照二元分类方式判别缺陷模块,而本文利用 $k$-NN 学习同时适用于分类及数值预测的特点,在缺陷预测过程中以数值预测方式计算未知软件模块在目标属性上的取值,再通过与学习产生的分类阈值进行比较以识别缺陷模块,分类阈值可根据需求自适应调整的特性为缺陷模块的判定提供较好的灵活性.

## 2　集成 $k$-NN 软件缺陷预测

通常 $k$-NN 方法预测不同实例时参照的近邻实例数量均为 $k$ 个,尽管该 $k$ 值可通过预先训练选择,但是固定 $k$ 值的预测方式忽略不同未知实例的个性特征,难以保证各实例均获得理想的预测精度,总体性能不理想. 因此,本文将 Boosting 集成方法及属性选择过程引入 $k$-NN 软件缺陷预测,试图通过该方式有效改善 $k$-NN 学习的性能. Boosting 是一种独立于特定算法的弱学习器提升方法,以多个不同弱学习器的集成提升单一学习器的预测精度. 本节首先给出基于 Boosting 的 $k$-NN 预测模型,接着以该模型为基础就集成 $k$-NN 软件缺陷预测算法的各个组成部分进行详细分析.

### 2.1　集成 $k$-NN 预测模型

与单一 $k$ 值的 $k$-NN 方法相对应,如果依据未知实例的不同特性以不同数量的近邻作为参照,并考虑不同属性在实例临近性度量中的不同作用,则预测过程就有可能兼顾各未知实例的个性化特征并对预测过程做出自适应的调整,总体预测精度的提升也就成为可能. 显然,固定 $k$ 值的预测方式是难以实

现此目标的，传统的 $k$-NN 方法仅能在一个原始数据集上训练产生一个预测器，虽然可通过分割数据集的方式生成多个训练子集，但对基于实例学习的 $k$-NN 方法而言，这样的处理方式将会使得参照样本过少而不利于近邻的合理搜索. 为了实现自适应的 $k$-NN 预测，需要若干个互不相同的 $k$-NN 预测器构成的学习器集合，Boosting 集成方法可较好地解决 $k$-NN 算法单一 $k$ 值预测方式的不足.

Boosting 是一种具有数据集实例权重更新机制的集成方法[15]. 该方法首先在原始数据集上依照实例权重分布抽样生成一个训练集并建立一个基本学习模型，然后以该模型对原始数据集中各实例的预测误差为基础调整实例权重，那些预测精度较差的实例将被赋予更高的权重，这样后续的抽样和训练过程将集中于这些较难实例的学习，如此反复，即可生成一个基本学习模型集合. 由于 Boosting 是一种算法无关的集成理论，因此可适应于不同的弱学习器学习算法.

以 Boosting 理论为基础，图 1 给出集成 $k$-NN 软件缺陷预测模型，该模型由一系列基本 $k$-NN 预测器构成，每个预测器由以下两个步骤建立. 1) 从原始数据集 $D$ 中经 Bootstrap 抽样生成一个新的训练集 $D_t$，$t=1,2,\cdots,T$; 2) 在 $D_t$ 上建立基本 $k$-NN 预测器 $h_t$. 由于 Bootstrap 抽样是有放回抽样，因此各抽样数据集 $D_t$ 的大小与原始数据集 $D$ 一致，但部分实例可能会重复出现于同一抽样集中，而另外的一些实例则不在某个抽样集中出现. 训练过程总是倾向于那些重复的实例，基本预测器 $h_t$ 应使 $D_t$ 中的实例获得尽可能理想的总体预测精度. Bootstrap 抽样迭代进行 $T$ 次后即可生成 $T$ 个基本 $k$-NN 预测器，各抽样数据集的差异性是 $T$ 个基本预测器对不同特性未知实例适应能力的重要保证.

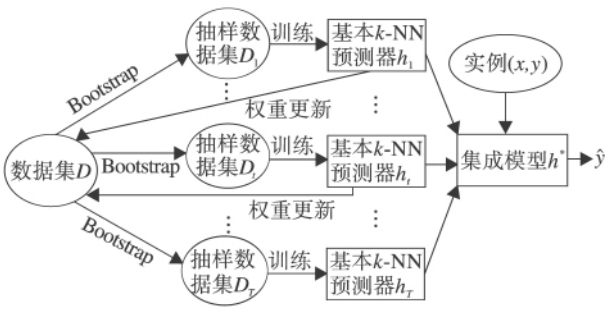


**图 1 集成 $k$-NN 软件缺陷预测模型**

Fig. 1 Ensemble model for $k$-NN defect prediction

如图 1 所示，我们以值对$(x,y)$表示一个实例，即软件模块，其中 $x$ 为模块在各个软件度量元上的取值，$y$ 为目标属性值，集成模型 $h^*$ 对实例$(x,y)$的预测结果为 $\hat{y}$. 通常，软件缺陷数据集在目标属性上的取值包括模块所含缺陷数量及是否为缺陷模块的二元标示(True/False) 两种情况. 本文集成 $k$-NN 预测模型以各实例目标属性取值为数值方式进行训练预测，对于目标属性为二元标示的数据集预先将其转换为数值 1 或 0.

在集成方法中，各基本预测器输出结果的集成策略是影响集成模型预测精度的重要因素，应用中可采用不同方式实现基本预测值的集成，如均值、中位数、加权和等. 考虑到缺陷预测数据分布的倾斜性及中位数对离群点数据的不敏感性等因素，集成 $k$-NN 预测模型以各基本预测值的中位数作为最终的集成预测结果. 不可避免的，集成预测方法较之单一学习器需要更多开销，不过对于预测精度的提升而言，这样的付出是值得的.

### 2.2 基本 $k$-NN 预测器

每个基本 $k$-NN 预测器均由两个参数描述: 1 个合理的 $k$ 值及相应的属性子集，训练过程以最小化总体误差的方式筛选出这两个参数. 具体的，误差可通过各种绝对或相对度量方式计算. 对于软件缺陷预测而言，由于缺陷通常仅分布于少量模块中，其余模块均为不含缺陷的正常模块，因此数据集中大部分实例在目标属性 $y$ 上的取值为 0，这使得相对误差难以计算. 在此情况下，采用平均绝对误差度量每个 $k$ 值及相应属性子集在抽样集上的预测性能. 值得注意的是，通常以最小化平均绝对误差为条件选择模型参数时可能会出现偏好极端实例的情况. 例如，对于个别具有很大 $y$ 值的实例，如果某参数恰好可准确预测此类实例，则在计算平均绝对误差时其降幅就会较显著，该组 $k$ 值及属性子集被选为模型参数的可能性也会较高，但实际情况可能仅仅是个别实例获得理想的预测精度，这并不符合参数选择的初衷. 我们更希望选定的模型参数应使训练集获得最理想的总体预测精度而不仅仅是实现个别实例的准确预测. 集成 $k$-NN 预测可在一定程度上避免该现象的出现. 训练集中因 Bootstrap 抽样产生的重复实例将具有相同的预测误差，此外，特征相似的实例其预测误差也较接近，这些实例在参数学习过程中将会影响平均绝对误差的计算，同时也会降低那些目标属性值较大的实例在平均绝对误差计算中的干扰，最终选择的基本预测器参数往往会在这些实例上取得较好的预测精度，这亦是 Boosting 方法的目的所在. 实例权重的不断更新保证了抽样数据集的

多样性,差异化的训练数据集使得各基本 $k$-NN 预测器的参数不尽相同,于是更多实例获得准确预测的可能性也就随之增加.

一般情况下,Boosting 方法仅适用于不稳定学习算法的性能提升,即那些训练集微小变动即可导致学习器输出结果及预测精度显著变化的算法. 然而,$k$-NN 通常被认为是一种稳定的学习算法,难以通过集成理论得到改进. 因此,集成 $k$-NN 预测中引入属性选择过程以扰动训练集的稳定性,使其更适于集成理论应用. 同时,合理的属性选择也有助于实例间临近性的准确度量并降低距离计算开销. 包装(Wrapper)和过滤(Filter)是两类普遍使用的属性选择方法. 包装方法选出的属性子集通常具有更好的预测精度,不过其计算复杂度也较高. 过滤方法的属性选择过程独立于学习算法,其精度较差但开销较小. 综合考虑性能及复杂度因素,基本 $k$-NN 预测器训练中采用与包装方法中的典型方法——后向消去法类似但复杂度显著降低的随机方法作为其属性子集选择方式. 后向消去方法需要逐一验证去除每个属性后算法的精度改进情况,并据此选择一个最佳的属性子集作为下一次迭代选择过程的属性全集,这是导致后向消去复杂度高的主要原因. 随机子集选择方法每次随机生成一个属性子集,如果学习算法在该子集上的性能优于产生当前子集的属性全集,则该属性子集作为下一轮迭代的属性全集并在此基础上继续生成新的随机子集. 反之,则选择过程中止. 对于含有 $d$ 个属性的数据集,可生成一个 $[0,1]$ 区间内的 $d$ 元随机向量,向量中各元素与 $d$ 个属性一一对应,当某分量随机数大于 0.5 时,对应属性被选入属性子集,否则相应属性被拒绝. 通过该方式即可快速生成一个属性子集. 后向消去方法为选出一个属性子集需要进行 $d$ 次验证,而随机子集方法只需一次验证即可. 虽然随机选择方法的属性子集每次都是根据概率随机生成,但该方法的迭代选择过程可在一定程度上保证选出性能较好的属性子集. 其总体性能介于过滤方法及后向消去方法之间.

近邻搜索空间的选择是基本 $k$-NN 预测器训练中的另一关键问题. 对于积极学习方法而言,基本学习器在抽样集上以留一法训练即可,然而该方式对基于实例学习的 $k$-NN 方法却是不可行的,原因在于抽样集中的大量重复实例. 如果采用类似积极学习算法的方式进行训练,基本 $k$-NN 预测器将在空间 $D_t-(\boldsymbol{x},y)$ 中搜索实例 $(\boldsymbol{x},y)\in D_t$ 的近邻,在此情况下,由于 $D_t$ 中重复实例的存在使得选择 $k$ 值为 1 即可准确预测这些实例,从而显著降低总体误差,但是如此产生的 $k=1$ 的选择由于失去泛化能力因而在实际预测应用中是没有意义的. 为避免该问题,基本 $k$-NN 预测器搜索近邻的空间应调整为 $D-(\boldsymbol{x},y)$.

令 $(\boldsymbol{x}_r,y_r)$ 表示抽样集 $D_t$ 中的第 $r,r=1,2,\cdots,n,n=|D_t|$,个实例,而 $d_j$ 为该实例与其第 $j,j=1,2,\cdots,k$,个近邻间的欧氏距离,则 $y_r$ 的预测值

$$\hat{y}_r=\sum_{j=1}^{k}w_jy_j$$

及其绝对误差 $e_r=\mathrm{abs}(y_r-\hat{y}_r)$,其中 $w_j$ 表示第 $j$ 个近邻的权重:

$$w_j=\frac{\frac{1}{d_j}}{\sum_{j=1}^{k}\frac{1}{d_j}}.$$

于是,当前 $k$ 值及属性子集在抽样训练集 $D_t$ 上的平均绝对误差:

$$\varepsilon=\frac{1}{n}\sum_{r=1}^{n}e_r.$$

产生最小平均绝对误差的 $k$ 值及属性子集将被指定为 $D_t$ 上的基本 $k$-NN 预测器 $h_t$ 的模型参数. 除平均绝对误差外,均方误差和均方根误差等绝对误差度量方式也可用作参数选择的标准,不过这两种方法对绝对误差进行平方计算的处理方式存在突出误差较大实例的可能性.

### 2.3　实例权重更新

基本预测器 $h_t$ 建立后即可根据其预测精度调整原始数据集 $D$ 中各实例权重. 由于下一基本预测器 $h_{t+1}$ 的训练集 $D_{t+1}$ 参照调整后的实例权重由 $D$ 中抽样产生,因此合理的权重更新对于 $h_{t+1}$ 的训练尤为重要.

当 Boosting 方法应用于分类问题时,其分类结果无非包括分类正确或错误两种情况,统计两种类别的实例数即可直接评估分类器的精度,因此其权重更新机制也相对简单. 然而,由于数值预测的精度无法像分类器评估那样进行,因此集成方法在数值预测中的权重更新过程将更为复杂. 在进行加权 Bootstrap 抽样之前必须给出一种适宜于数值预测的 Boosting 实例权重更新方法. 一种理论上可行的方法是将数值预测映射为无限多个二元分类问题,这些二元分类用于预测值是否大于或小于真实值的判断,随后即可将 Boosting 应用于这些二元分类的改进. 尽管已有学者提出不同类型的 Boosting 数值预测方法,但其中大部分都因计算复杂度过高而导致适用性较差. 集成 $k$-NN 软件缺陷预测以 Drucker[16] 提出的一种轻量级 Boosting 回归方法为

基础实现实例权重的动态调整.

具体的,首先采用损失函数将绝对误差映射至[0,1]区间内,于是数据集 $D$ 中各实例的权重即可按照加权平均损失更新. 对于 $(x,y)\in D$ 的损失:

$$L=\frac{|\hat{y}-y|}{M}, \tag{1}$$

其中,$M=\sup(\hat{y}_i-y_i)$,$i=1,2,\cdots,n$,即为当前基本预测器 $h_t$ 预测 $D$ 中各实例时的最大绝对误差. 除式(1)外,区间[0,1]内的其它任何单调增函数都可用作损失函数. 在此基础上,原始数据集 $D$ 中的第 $i$ 个实例权重依 $h_t$ 更新如下:

$$\bar{L}_t=\sum_{i=1}^{n}w_t(i)L_t(i), \tag{2}$$

其中,$w_t(i)$ 和 $L_t(i)$ 分别表示 $D$ 中第 $i$ 个实例$(x_i,y_i)$在第 $t$ 次迭代训练过程中的权重及基本预测器 $h_t$ 预测该实例时的损失,而 $\bar{L}_t$ 则是 $h_t$ 预测 $D$ 中各实例的平均加权损失. 在此基础上,给出 $h_t$ 的置信度:

$$\beta_t=\frac{\bar{L}_t}{1-\bar{L}_t}, \tag{3}$$

小的 $\beta_t$ 值意味着更低的加权平均误差及更好的置信度. 根据总体置信度 $\beta_t$ 及损失分别调整 $D$ 中各实例权重

$$v_{t+1}(i)=w_t(i)\beta_t^{1-L_t(i)}, \tag{4}$$

进行归一化处理,得

$$w_{t+1}(i)=\frac{v_{t+1}(i)}{\sum_{i=1}^{n}v_{t+1}(i)}, \tag{5}$$

根据新的权重向量 $w_{t+1}$ 抽样即可生成下一个训练数据集 $D_{t+1}$.

### 2.4 分类阈值

由于软件缺陷分布的不均衡性,当采用 $k$-NN 方法进行预测时,正常模块不可避免的将会出现在一些未知实例的参照近邻中. 这些模块对应的数据集实例在目标属性上取值为零,使得 $k$-NN 方法以距离加权和计算得到的预测值往往小于实际值. 近邻中正常模块数越多,则基本预测器预测值偏小的可能性也就越高,这将导致集成预测结果也偏小. 集成 $k$-NN 预测方法如果直接按照集成预测结果的绝对数值大小判别新的软件模块是否为缺陷模块,则将有相当一部分模块可能由于预测值过小而被误识别为无缺陷的,从而影响预测效果. 为解决该问题,本节给出一种自适应的学习方法,该方法从训练集各实例的集成预测结果中选出一个值作为区分是否为缺陷模块的阈值. 集成预测结果小于该阈值的模块将被标记为正常模块,否则为缺陷模块. 尽管集成 $k$-NN 是对缺陷分布进行预测,输出结果为二元标示,但在预测过程中并未直接采用 $k$-NN 分类方法,而是以数值预测方式进行,对于标示为 True/False 的数据集预先将其转换为 1/0 数值. 该方式较之直接采用 $k$-NN 分类进行预测增加阈值自适应学习的开销,不过集成预测结果的相对数值却可对模块隐含的缺陷程度提供度量参考,对于测试资源的分配具有一定的价值.

阈值学习过程以 Menzies 等[3] 提出的软件缺陷预测评价指标作为阈值选择依据. 由于该文已在软件缺陷预测领域引起广泛关注,具有较好的代表性及实用性,因此本文以此作为参照基准. 此外,本文还将就集成 $k$-NN 缺陷预测与该文所提基准预测方法的性能差异进行标准数据集上的实验验证.

通常,可给出以下 4 种二元分类结果正确性统计方式:

1) $TP$ (True Positive):被正确分类的肯定实例数;

2) $TN$ (True Negative):被正确分类的否定实例数;

3) $FP$(False Positive):被误分类为肯定实例的否定实例数;

4) $FN$(False Negative):被误分类为否定实例的肯定实例数.

对缺陷预测而言,当某个正常模块被误识别为缺陷模块时,该分类即是一个错误的肯定;反之,当某个缺陷模块被识别为正常时该分类则是错误的否定. 根据上述定义,Menzies 提出衡量软件缺陷预测精度的度量标准 $pd$ 及 $pf$,正确识别一个缺陷模块的概率 $pd$(Probability of Detection)即为正确检测出的缺陷模块数与真实的缺陷模块数之比:

$$pd=\frac{TP}{TP+FN}. \tag{6}$$

该概率值通常也被称为召回率(Recall). 另一方面,分类时的误报率 $pf$(Probability of False Alarm)则为错误识别的缺陷模块数与实际的正常模块数之比:

$$pf=\frac{FP}{FP+TN}. \tag{7}$$

我们总是希望能找到既可最大化 $pd$ 又能最小化 $pf$ 的分类方法,不过在实际中这两个目标却是难以同时实现的. 提高 $pd$ 意味着需要正确检测出更多的缺陷(Positive)模块,然而当有更多的模块被识别为 positive 时,其中不可避免的也会包括一些被误分类的正常(Negative)模块,于是 $pf$ 也随之升高. 反

之,为了降低 $pf$,势必也要以放弃检测出更多的缺陷模块为代价. Menzies 定义另一个能平衡 $pd$ 和 $pf$ 间关系的度量 $bal$:

$$bal = 1 - \frac{\sqrt{(0 - pf)^2 + (1 - pd)^2}}{\sqrt{2}}. \quad (8)$$

度量 $bal$ 实际上表示某个值对$\langle pf, pd \rangle$与 $ROC$ 曲线甜蜜点(Sweet Spot)$\langle pf = 0, pd = 1 \rangle$间的临近性. 为便于表示和理解,以最大可能距离$\sqrt{2}$对两点间欧氏距离进行标准化并取"1"与该距离之差为 $bal$,这样当 $bal$ 值越大时就表示$\langle pf, pd \rangle$与甜蜜点间的距离越近,分类器性能越好. 反之则越差. 集成 $k$-NN 预测以 $bal$ 作为分类阈值选择标准,具体过程包括以下步骤.

step 1　如果训练集 $D$ 中实例目标属性值为数值,则需对 $D$ 进行预处理. 目标属性取值为零的实例将被识别为正常模块,其标记改为 $negative$($no$),取值大于零的模块为缺陷模块,将其标记改为 $positive$($yes$).

step 2　依次以 $D$ 中各实例的集成预测结果作为划分标准,将各实例分为两类,集成预测结果小于该标准值的实例被分类为 $negative$ 模块,否则即为 $positive$.

step 3　以 step 1 中各实例真实类别为参照,比较各实例分类结果正确性并分别统计 $TP$、$TN$、$FP$、$FN$.

step 4　依次以各训练实例集成预测值作为分类标准,按照式(6) ~ (8) 计算其 $bal$ 值. 与最大 $bal$ 相对应的训练实例集成预测值即为选定的分类阈值.

实际上,选择阈值的标准并不惟一,上述方法选出的阈值可检测出最多的缺陷模块,但也很可能会随之带来较高的 $pf$. 因此,对于那些不希望有过高误报率的应用而言,以最大化 $bal$ 为阈值选择标准并不见得能够取得理想的预期效果. 此时,可在阈值选择过程中加入适当的约束,例如要求选中的 $bal$ 所对应的 $pf$ 值不超过以各集成预测结果分别作为划分标准时生成的全部 $pf$ 值的中位数或均值. 如果某个应用环境能接受较高的 $pf$ 或预测时产生的误报率本身已较低,则采用最大化 $bal$ 的方法不失为一种好的选择.

### 2.5　算法步骤

本文对集成 $k$-NN 软件缺陷预测算法中具有随机属性选择的基本学习器训练,实例权重更新,分类阈值自适应学习等重要环节进行详细介绍,下面给出该算法的完整实现.

**算法**　基于 Boosting 的集成 $k$-NN 软件缺陷预测算法

**输入**　训练数据集 $D$,未知实例$(x, y)$

**输出**　实例$(x, y)$的预测值 $\hat{y}$,分类阈值 $thr$,分类标记 $l(x)$

将各数值型属性归一化至[0, 1]区间;
初始化训练集 $D$ 中各实例权重:
$w_1(i) = \frac{1}{n}, i = 1, 2, \cdots, n, n = |D|$;
for $t = 1, 2, \cdots T$;　// 开始建立基本 $k$-NN 预测器
根据权重向量 $w_t$ 对数据集 $D$ 进行 Bootstrap 抽样,生成训练集 $D_t$, $|D_t| = n$;
for $k = 1, 2, \cdots, K_{\max}$　// 训练基本预测器 $h_t$
以 $k$ 个近邻为参照依次预测 $D_t$ 中各实例,记平均绝对误差为 $e$;
$D_t' = D_t$;　// 开始属性选择
$m = MaxVal$;　// $MaxVal$ 为预先给定的任意极大数值
while $e < m$
$m = e$;
从 $D_t'$ 当前属性全集 $U$ 中随机选择一个属性子集 $U'$;
将 $D_t'$ 投影至属性子集 $U'$: $D_t' = D_t(U')$;
以 $k$ 个近邻为参照依次预测 $D_t'$ 中各实例,记平均绝对误差为 $e$;
end;　// end of while
$\varepsilon(k) = m$;
$A(k) = U$;　// $U$ 为当前属性全集
end for;
$K_t = \arg\min_k(e(k))$, $k = 1, 2, \cdots, K_{\max}$;
//$K_t$ 及对应的属性子集 $A(k)$ 即为 $h_t$ 的参数
根据式(1) 计算 $D$ 中各实例的损失 $L_t(i)$, $i = 1, 2, \cdots, |D|$;
根据式(2) 计算加权平均损失 $\bar{L}_t$;
根据式(3) ~ 式(5) 调整 $D$ 中各实例权重并形成新的权重向量 $w_{t+1}$;
end for　//$T$ 个基本 $k$-NN 预测器 $h_1, h_2, \cdots, h_T$ 已建立
采用集成 $k$-NN 模型以留一法依次预测 $D$ 中各实例,记 $\hat{y}_i$ 为第 $i$ 个实例$(x_i, y_i)$ ($i = 1, 2, \cdots, |D|$) 的集成预测结果;
for $i = 1, 2, \cdots, |D|$　// 声明 $D$ 中各实例的真实标记
if $y_i > 0$ *then* $l(x_i) = 1$ else $l(x_i) = 0$;
end for
for $i = 1, 2, \cdots, |D|$　// 开始阈值选择
以 $\hat{y}_i$ 为阈值根据式(6) ~ 式(8) 分别计算 $pd(i)$、$pf(i)$ 及 $bal(i)$;
end for
对各 $bal$ 值进行排序;
选择最大 $bal$ 对应的集成预测值为阈值 $thr$;
以集成 $k$-NN 模型预测未知实例$(x, y)$:
$$\hat{y} = \text{median}(\hat{y}_1, \hat{y}_2, \cdots, \hat{y}_T);$$
// $\hat{y}_1, \hat{y}_2, \cdots, \hat{y}_T$ 分别为基本预测器 $h_1, h_2, \cdots, h_T$ 的预测输出

33 if $\hat{y} \geqslant thr$ then $l(x) = 1$ else $l(x) = 0$.

该算法主要由集成预测模型训练及未知实例预测两部分构成. 算法首先对数值属性及实例权重进行初始化,接下来3 ~ 22行是建立$T$个基本$k$-NN预测器的过程. 第5行中的$K_{\max}$表示需要考虑的最大参照近邻数量,该值可在算法开始运行前预先设定. 由于属性子集每次均为随机生成,为避免属性子集无法改进预测精度的情况,算法第6行在属性选择开始前首先计算属性全集上的平均绝对误差,当该误差小于属性全集上的平均绝对误差时,选择过程继续进行,否则表明随机选择的属性子集未能改进预测精度,当前$k$值的属性选择结束. 属性全集上的平均绝对误差总是小于第8行中引入的任意极大值$MaxVal$,因此必然可进入10 ~ 13行的属性选择过程. 从第24行开始算法进行分类阈值的学习,首先给出各模块实例的真实标记,$l(\boldsymbol{x}_i)$即为第$i$个实例的标记,$l(\boldsymbol{x}_i) = 1$表示该模块为缺陷模块,而$l(\boldsymbol{x}_i) = 0$表示该模块为正常模块. 在此基础上,27 ~ 31行依次以$D$中每个实例的集成预测结果作为备选阈值计算$bal$,并选择最大值对应的集成预测结果作为阈值$thr$. 第32 ~ 33行预测未知实例$(\boldsymbol{x}, y)$并根据$thr$判断其类别.

## 3 实验和结果分析

本节对集成$k$-NN软件缺陷预测算法的性能进行实验验证,并与其它缺陷预测方法的精度进行实验分析对比. 首先对实验所用方法进行说明,接下来对实验结果进行分析.

### 3.1 实验设置

本节实验所用数据集分别来自于软件缺陷预测研究中广泛使用的NASA MDP repository (http://mdp.ivv.nasa.gov)及PROMISE code repository (http://promisedata.org/?cat=4)数据库. 美国国家航空航天局提供的开放软件缺陷数据库平台NASA MDP repository,收集北美地区不同组织开发的4个应用系统的项目数据. 这些系统分别采用C语言或Java语言实现,各数据集目标属性取值为模块中的缺陷数. 表1给出与数据集规模有关的代码行总数、缺陷总数、模块数、缺陷模块数及其所占比率等具体描述信息. 其中模块数即为各数据集中的实例数量,每个实例表示一个具体的软件模块,可看到kc4数据集的模块实例数较少但缺陷模块比率较高,其它数据集所含实例相对较多且缺陷模块比例接近.

表1 实验中使用的NASA MDP数据集

Table 1 NASA MDP datasets used in experiment

| 数据集 | LOC | 缺陷总数 | 模块总数 | 缺陷模块数 | 缺陷模块占比/% |
|---|---|---|---|---|---|
| cm1 | 16903 | 70 | 505 | 48 | 9.50 |
| kc3 | 7749 | 101 | 458 | 43 | 9.39 |
| kc4 | 25436 | 253 | 125 | 61 | 48.80 |
| mw1 | 8341 | 37 | 403 | 31 | 7.69 |
| pc1 | 25922 | 139 | 1107 | 76 | 6.87 |
| pc3 | 36473 | 259 | 1563 | 160 | 10.24 |
| pc4 | 30055 | 367 | 1458 | 178 | 12.21 |

初始情况下以上各数据集均由43个属性构成,这些属性中除了模块标示符(Module)、缺陷数量(error_count)及缺陷密度(error_density)3个属性外都是依照Halsted、LOC、McCabe等软件规模及复杂度度量方法生成. 由于模块标示符属性仅用于区分每个模块且取值各不相同,这样的属性将会造成预测算法训练过程中的过拟合现象,因此在数据集预处理时删除该属性. 此外,error_count属性即为每个模块中的具体缺陷数量,以该属性作为目标属性,error_density为error_count的导出属性,因此一并删除. 经过预处理后的NASA MDP数据集包含40个软件度量描述属性及1个目标属性. 关于数据集各属性的具体构成及详细说明可参阅数据集所在网站.

美国西弗吉尼亚大学计算机科学系Boetticher等共同开发一个用于共享软件工程数据的PROMISE(Predictor Models in Software Engineering)网站,提供可用于软件工程研究的多种软件项目数据. 本节实验所用的AR系列数据集即来自于其中的PROMISE code repository. 该系列数据集源自土耳其某白色家电制造商采用C语言开发的嵌入式软件系统. 由于AR是二元分类数据集,在采用集成$k$-NN算法预测前需预先对目标属性进行一定的转换. AR数据集的目标属性仅用True或False表示缺陷模块及正常模块,我们统一将目标属性上的值"True"转换为1,而将"False"改为0. 通过分类阈值的比较可将集成预测结果转化为分类结果,该处理方式与一般的分类方法的区别在于可通过调整阈值大小灵活改变$pd$、$pf$及$bal$度量. 表2给出各AR数据集的大小规模,因该系列数据集为分类数据集,因此表中未给出缺陷总数信息. 由表2可看出,各数据集的规模均明显小于NASA MDP数据集,但是除AR1外,各数据集的缺陷模块占比都相对较高.

表 2　实验中使用的 PROMISE AR 数据集

Table 2　PROMISE AR datasets used in experiment

| 数据集 | LOC | 模块总数 | 缺陷模块数 | 缺陷模块占比/% |
|---|---|---|---|---|
| AR1 | 2467 | 121 | 9 | 7.44 |
| AR3 | 5624 | 63 | 8 | 12.70 |
| AR4 | 9196 | 107 | 20 | 18.69 |
| AR5 | 2732 | 36 | 8 | 22.22 |
| AR6 | 2078 | 101 | 15 | 14.85 |

AR 系列数据集均由 29 个描述属性及 1 个目标属性构成,这些属性同样通过 Halstead、LOC 及 McCabe 等软件度量方法产生. 关于属性的构成及具体描述可参阅数据集所在网站.

### 3.2　实验方法

实验所用对比方法分别为朴素贝叶斯( Naïve Bayes) 分类、J48 决策树、基于规则的 OneR 方法、支持向量机( SVM) 及标准 $k$-NN 分类器. 其中前 3 种方法为 Menzies [3] 等提出的基准方法. 支持向量机方法为 Elish [13] 等采用的缺陷预测方法. 为了进一步分析 Boosting 集成理论对 $k$-NN 缺陷预测方法的性能提升作用,本实验还同时实现基本的 $k$-NN 分类器.

为了与上述参照方法实验结果进行对比,采用完全相同的实验过程,集成 $k$-NN 预测算法及各参照方法在每个缺陷数据集上以 10 折交叉验证方式重复进行 10 次. 实验主要针对集成 $k$-NN 预测及参照方法在各数据集上的 $pd$、$pf$ 及 $bal$ 度量指标进行比较,各数据集 10 次实验结果的均值将作为该数据集上的最终实验结果. 每次实验过程中集成 $k$-NN 方法均建立 6 个基本 $k$-NN 预测器.

### 3.3　实验结果分析

本节将分别对 NASA MDP 及 PROMISE AR 这 2 种数据集上的实验结果进行分析.

#### 3.3.1　NASA MDP 数据集实验结果

表 3 为 NASA MDP 数据集实验结果,其中,B$k$-NN 即为基于 Boosting 的集成 $k$-NN 预测方法. 可看到,B$k$-NN 方法总体上取得最为理想的预测效果,在 cm1、kc3、mw1、pc1、pc3 数据集上的 $bal$ 度量均优于各对比方法,且在表 3 最后给出的平均结果中,$bal$ 度量亦明显优于这些方法. 作为实验对比的 5 种参照方法中,朴素贝叶斯方法的总体预测效果相对最好,并在 pc4 数据集上取得最佳的 $bal$ 度量. J48 决策树方法在 kc4 数据集上的预测精度最为理想,而在其它缺陷数据集上的性能则明显劣于 B$k$-NN 及朴素贝叶斯方法,该现象与 kc4 数据集中缺陷分布比例较高有关. OneR 方法虽然可获得非常低的误报率,但同时可发现的缺陷模块也非常少,这导致该方法的 $pf$ 及 $pd$ 度量都很低,缺乏实用性.

需要说明的是,文献 [13] 对基于支持向量机的缺陷预测方法进行研究,并给出部分 NASA MDP 数据集上的

$$Accuracy=\frac{TP+TN}{TP+TN+FP+FN},$$

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

等实验结果. 虽然该文中支持向量机方法的实验结果较理想,但由于其针对数据集中的正常模块而非缺陷模块分类情况进行统计评估分类器性能,因此该评价方式对缺陷预测而言是不合理的. 缺陷分布的不均衡性使得数据集中大量存在的是无缺陷的正常模块,而识别这些模块对及时修正软件中隐含的缺陷是无价值的,缺陷预测的目的仅在于准确找出那些含有缺陷的少量模块,因此对缺陷预测算法的评价指标应针对缺陷模块进行,而非文献 [13] 的处理方式. 此外,$pf$ 是评价缺陷预测算法产生误报情况的重要指标,文献 [13] 忽略此度量,未能考虑误报率 $pf$ 对算法性能的影响. 基于上述分析,本文实验采用的支持向量机方法针对缺陷模块统计度量指标,并同时采用 $pf$ 度量. 由表 3 可看到,支持向量机方法的 $pd$ 及 $pf$ 度量值都非常低,因此难以发现更多的缺陷模块,该方法在本实验的各种对比方法中效果是最差的,其对不平衡缺陷数据分布的适应性不足应是造成该现象的主要原因.

比较表 3 中 B$k$-NN 预测与标准 $k$-NN 分类器的实验结果可知,基于 Boosting 的集成 $k$-NN 缺陷预测方法较之标准的 $k$-NN 分类有较明显提升,这表明集成理论及本文中给出的分类阈值自适应学习方法对 $k$-NN 分类具有较好的提升作用.

由于朴素贝叶斯在各参照方法中预测性能最佳,因此以下就 B$k$-NN 与朴素贝叶斯方法的实验结果加以具体分析. 可看到,与朴素贝叶斯方法相比,集成 $k$-NN 较有效地降低除 pc4 外各数据集的误报率 $pf$,进而提升这些数据集上的 $bal$ 度量. 特别是,集成 $k$-NN 在 pc1 数据集上的预测精度改善最为显著,不仅有效提高 $pd$,同时还降低 $pf$,这使得 $bal$ 度量也随之获得了 15.47% 的显著提升. 在 kc3 数据集上,集成 $k$-NN 与朴素贝叶斯方法的 $pd$ 值非常接近,但由于可有效降低 $pf$,因此集成 $k$-NN 预测仍能较之朴素贝叶斯显著提升 $bal$. 此外,集成 $k$-NN 在 cm1、kc4、mw1、pc3 等数据集上的 $pd$ 值虽然略低于朴素贝叶斯方法,但其误报率也明显低于后者,因此

表 3 NASA MDP 数据集实验结果

Table 3 Experimental results on NASA MDP datasets

| 数据集 | 度量 | B$k$-NN | Naïve Bayes | J48 | OneR | SVM | $k$-NN |
|---|---|---|---|---|---|---|---|
| cm1 | *pd* | 0.746 | 0.770 | 0.152 | 0.086 | 0.041 | 0.193 |
| | *pf* | 0.291 | 0.322 | 0.051 | 0.023 | 0.002 | 0.096 |
| | *bal* | **0.726** | 0.695 | 0.399 | 0.353 | 0.322 | 0.424 |
| kc3 | *pd* | 0.763 | 0.765 | 0.221 | 0.113 | 0 | 0.346 |
| | *pf* | 0.241 | 0.327 | 0.043 | 0.016 | 0 | 0.050 |
| | *bal* | **0.759** | 0.697 | 0.447 | 0.372 | 0.293 | 0.535 |
| kc4 | *pd* | 0.738 | 0.769 | 0.916 | 0.757 | 0.115 | 0.736 |
| | *pf* | 0.294 | 0.364 | 0.321 | 0.317 | 0.063 | 0.335 |
| | *bal* | 0.720 | 0.681 | **0.745** | 0.692 | 0.370 | 0.674 |
| mw1 | *pd* | 0.587 | 0.650 | 0.218 | 0.080 | 0 | 0.298 |
| | *pf* | 0.190 | 0.264 | 0.021 | 0.021 | 0.006 | 0.080 |
| | *bal* | **0.677** | 0.665 | 0.445 | 0.349 | 0.293 | 0.499 |
| pc1 | pd | 0.651 | 0.586 | 0.249 | 0.070 | 0.104 | 0.377 |
| | *pf* | 0.173 | 0.269 | 0.020 | 0.009 | 0.005 | 0.042 |
| | *bal* | **0.724** | 0.627 | 0.468 | 0.343 | 0.367 | 0.558 |
| pc3 | *pd* | 0.708 | 0.794 | 0.202 | 0.086 | 0.070 | 0.301 |
| | *pf* | 0.209 | 0.347 | 0.042 | 0.011 | 0.002 | 0.058 |
| | *bal* | **0.746** | 0.710 | 0.435 | 0.354 | 0.342 | 0.503 |
| pc4 | *pd* | 0.830 | 0.859 | 0.942 | 0.980 | 0.022 | 0.926 |
| | *pf* | 0.210 | 0.201 | 0.500 | 0.729 | 0.008 | 0.536 |
| | *bal* | 0.807 | **0.822** | 0.643 | 0.484 | 0.308 | 0.617 |
| 平均值 | *pd* | 0.718 | 0.742 | 0.414 | 0.310 | 0.050 | 0.454 |
| | *pf* | 0.230 | 0.299 | 0.143 | 0.161 | 0.012 | 0.171 |
| | *bal* | **0.737** | 0.700 | 0.512 | 0.421 | 0.328 | 0.544 |

总体上预测性能仍优于朴素贝叶斯方法. 最后,集成 $k$-NN 在 pc4 数据集上的 *pd*、*pf* 及 *bal* 度量均不如朴素贝叶斯方法,这可能与该数据集缺陷模块的分布有关. 表 3 最后给出的各数据集平均结果表明集成 $k$-NN 缺陷预测的性能总体上优于朴素贝叶斯方法,而各数据集 *bal* 值间的单尾 $t$-检验也验证集成 $k$-NN 预测较之朴素贝叶斯方法的提升具有统计显著性 ($p=0.0159$).

实验还对集成 $k$-NN 方法训练过程中产生的各个属性子集进行统计,表 4 所示为各数据集基本预测器选择次数居前 3 位的属性,由于一些属性的出现频次相同,因此表中存在属性数多于 3 个的情况. 可以看到表中属性涉及 Halstead、LOC 及 McCabe 等不同度量方法,而且部分属性还在不同数据集中同时出现,这也从侧面说明集成 $k$-NN 预测方法中属性选择的一致性和有效性.

### 3.3.2 PROMISE AR 数据集实验结果

表 5 所示为集成 $k$-NN 预测及 5 种对比方法在 AR 系列数据集上关于 *pd*、*pf* 及 *bal* 度量的实验结果对比. 可看出,与 NASA MDP 数据集上的实验结果类似,5 种参照方法中,朴素贝叶斯方法的预测精度相对较为理想,而支持向量机方法缺乏对 AR 数据集的适应性,未能识别出各数据集中的缺陷模块,故各次实验中 *pd* 值均为 0. 而 J48 决策树、OneR、标准 $k$-NN 分类的预测性能介于朴素贝叶斯与支持向量机之间.

表 4 频繁选择的 NASA MDP 数据集属性

Table 4 Most often used attributes of NASA MDP datasets

| 数据集 | 属 性 |
|---|---|
| cm1 | num_unique_operators, design_density, percent_comments |
| kc3 | global_data density, num_unique_operators, decision_count |
| kc4 | global_data_density, multiple_condition_count, node_count, number_of_lines, loc_total |
| mw1 | global_data_density, loc_code_and_comment, loc_executable, branch_count, node_count, num_operators |
| pc1 | percent_comments, design_density, halstead_content, loc_blank |
| pc3 | normalized_cyclomatic_complexity, halstead_volume, node_count, num_unique_operators |
| pc4 | percent_comments, num_unique_operators, num_operators |

表 5　PROMISE AR 数据集实验结果

Table 5　Experimental results on PROMISE AR datasets

| 数据集 | 度量 | B$k$-NN | Naïve Bayes | J48 | OneR | SVM | $k$-NN |
|---|---|---|---|---|---|---|---|
| AR1 | *pd* | 0.574 | 0.400 | 0.110 | 0.030 | 0 | 0.290 |
| | *pf* | 0.228 | 0.286 | 0.033 | 0.026 | 0.015 | 0.053 |
| | *bal* | **0.657** | 0.428 | 0.369 | 0.310 | 0.292 | 0.482 |
| AR3 | *pd* | 0.714 | 0.600 | 0.430 | 0.310 | 0 | 0.330 |
| | *pf* | 0.068 | 0.110 | 0.056 | 0.037 | 0 | 0.060 |
| | *bal* | **0.791** | 0.661 | 0.579 | 0.504 | 0.293 | 0.513 |
| AR4 | *pd* | 0.686 | 0.650 | 0.385 | 0.330 | 0 | 0.295 |
| | *pf* | 0.199 | 0.235 | 0.094 | 0.076 | 0 | 0.119 |
| | *bal* | **0.735** | 0.664 | 0.538 | 0.511 | 0.293 | 0.486 |
| AR5 | *pd* | 0.821 | 0.700 | 0.490 | 0.570 | 0 | 0.500 |
| | *pf* | 0.133 | 0.133 | 0.133 | 0.108 | 0 | 0.185 |
| | *bal* | **0.839** | 0.733 | 0.582 | 0.651 | 0.293 | 0.571 |
| AR6 | *pd* | 0.625 | 0.300 | 0.110 | 0.095 | 0 | 0.315 |
| | *pf* | 0.288 | 0.161 | 0.047 | 0.025 | 0.025 | 0.078 |
| | *bal* | **0.662** | 0.470 | 0.366 | 0.358 | 0.292 | 0.497 |
| 平均值 | *pd* | 0.684 | 0.530 | 0.305 | 0.267 | 0 | 0.346 |
| | *pf* | 0.183 | 0.185 | 0.073 | 0.054 | 0.008 | 0.099 |
| | *bal* | **0.737** | 0.591 | 0.487 | 0.467 | 0.293 | 0.510 |

与朴素贝叶斯方法相比，集成 $k$-NN 在各数据集上均取得良好的预测效果，不仅极大提高 *pd* 及 *bal* 度量值，而且有效降低 AR1、AR3、AR4 上的误报率 *pf*. 虽然两种方法在 AR5 上的 *pf* 值相同，但集成 $k$-NN 却能准确预测出更多的缺陷模块. 此外，集成 $k$-NN 较之朴素贝叶斯方法将 AR6 数据集上的 *pd* 值提升一倍以上，尽管 *pf* 有所增加，但整体上仍极大提升 *bal* 指标. 与 NASA MDP 数据集实验结果类似，集成 $k$-NN 及朴素贝叶斯方法在各 AR 数据集上 *bal* 值单尾 $t$-检验结果( $p = 0.0036$) 表明集成 $k$-NN 对缺陷预测的提升同样具有统计显著性.

总体上，朴素贝叶斯方法在 NASA MDP 数据集上取得较好的预测效果，但在 AR 数据集上表现出的性能却不尽理想，表明朴素贝叶斯方法的适用范围具有一定的局限性，而集成 $k$-NN 预测在两类数据集上均具有较好的预测能力，表现出更好的适应性. 集成 $k$-NN 预测方法在 NASA MDP 及 PROMISE AR 数据集上取得的良好效果也进一步验证 Halstead、LOC、McCabe 等度量方法在软件缺陷预测中的有效性.

与 NASA MDP 数据集类似，表 6 给出集成 $k$-NN 方法在 PROMISE AR 数据集上训练时频繁使用的属性列表. 可看到，该系列数据集使用最多的属性主要集中于 Halstead 及 McCabe 度量中，尤其是操作符种类数( unique_operators) 在 AR4、AR5、及 AR6 数据集中对缺陷模块的判别有着至关重要的影响，而软件代码行规模度量( LOC) 对 AR 系列数据集的作用并不突出.

表 6　频繁选择的 PROMISE AR 数据集属性

Table 6　Most often used attributes of PROMISE AR datasets

| 数据集 | 属　性 |
|---|---|
| AR1 | total_operands, design_density, decision_density |
| AR3 | halstead_difficulty, halstead_time, branch_count |
| AR4 | unique_operators, halstead_vocabulary, halstead_effort, halstead_error |
| AR5 | unique_operators, cyclomatic_density, halstead_length |
| AR6 | halstead_volume, unique_operators, design_complexity, halstead_level, halstead_time |

## 4　结 束 语

本文将近邻学习方法应用于软件缺陷分布预测过程，给出基于 Boosting 的集成 $k$-NN 缺陷预测模型与算法. 算法首先通过加权抽样建立多个基本$k$-NN 学习器，各基本学习器独立给出其预测结果并以所有预测结果的中位数作为最终的集成预测结果. 集成理论避免传统 $k$-NN 方法基于单一 $k$ 值预测，总体预测精度难以保证的不足，有效提升 $k$-NN

算法的性能. 基本预测器训练过程中的属性选择功能改变 $k$-NN 学习的稳定性,有效扰动训练集,进一步增强集成 $k$-NN 的有效性. 自适应分类阈值是集成 $k$-NN 缺陷预测算法的另一特点. 由于数据集分布的不平衡性,集成 $k$-NN 预测结果总是具有偏小的趋势,因此在判别软件模块是否为缺陷模块时引入判别阈值的学习机制,集成 $k$-NN 算法根据基准度量指标从训练集各实例预测结果中选出一个作为识别缺陷模块的参照值,进行缺陷模块分布的预测. 该阈值的选择条件可根据实际应用环境的不同而做出相应的调整,具有较好的可用性和灵活性. 经过与其他学者所提基准缺陷分布预测方法的比较,集成 $k$-NN 方法在软件缺陷预测方面取得较好的效果. 如何改进缺陷模块分布稀疏性引起的基于实例的学习方式预测结果偏小的问题将是下一步的研究方向.

## 参 考 文 献


[1] Nikora A, Munson J. Developing Fault Predictors for Evolving Software Systems // Proc of the 9th International Software Metrics Symposium. Sydney, Australia, 2003: 338-350

[2] Nagappan N, Ball T. Static Analysis Tools as Early Indicators of Prerelease Defect Density // Proc of the 27th International Conference on Software Engineering. St. Louis, USA, 2005: 580-586

[3] Menzies T, Greenwald J, Frank A. Data Mining Static Code Attributes to Learn Defect Predictors. IEEE Trans on Software Engineering, 2007, 33(1): 2-13

[4] Lessmann S, Baesens B, Mues C, *et al*. Benchmarking Classification Models for Software Defect Prediction: A Proposed Framework and Novel Findings. IEEE Trans on Software Engineering, 2008, 34(4): 485-496

[5] Khoshgoftaar T M, Seliya N. Analogy-Based Practical Classification Rules for Software Quality Estimation. Empirical Software Engineering, 2003, 8(4): 325-350

[6] Emam K E, Benlarbi S, Goel N, *et al*. Comparing Case-Based Reasoning Classifiers for Predicting High Risk Software Components. Journal of Systems and Software, 2001, 55(3): 301-320

[7] Turhan B, Bener A. Analysis of Naïve Bayes' Assumptions on Software Fault Data: An Empirical Study. Data and Knowledge Engineering, 2009, 68(2): 278-290

[8] Khoshgoftaar T M, Allen E B, Hudepohl J P, *et al*. Application of Neural Networks to Software Quality Modeling of a Very Large Telecommunications System. IEEE Trans on Neural Networks, 1997, 8(4): 902-909

[9] Zheng Jun. Cost-Sensitive Boosting Neural Networks for Software Defect Prediction. Expert Systems with Applications, 2010, 37(6): 4537-4543

[10] Selby R W, Porter A A. Learning from Examples: Generation and Evaluation of Decision Trees for Software Resource Analysis. IEEE Trans on Software Engineering, 1988, 14(12): 1743-1757

[11] Guo Lan, Ma Yan, Cukic B, *et al*. Robust Prediction of Fault-Proneness by Random Forests // Proc of the 15th International Symposium on Software Reliability Engineering. Saint-Malo, France, 2004: 417-428

[12] Ceylan E, Kutlubay F O, Bener A B. Software Defect Identification Using Machine Learning Techniques // Proc of the 32nd Euromicro Conference on Software Engineering and Advanced Applications. Cavtat, Croatia, 2006: 240-246

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

[14] Shepperd M, Schofield C. Estimating Software Project Effort Using Analogies. IEEE Trans on Software Engineering, 1997, 23(11): 736-743

[15] Freund Y, Schapire R. Decision-Theoretic Generalization of On-line Learning and an Application to Boosting. Journal of Computer and System Sciences, 1997, 55(1): 119-139

[16] Drucker H. Improving Regressors Using Boosting Techniques // Proc of the 14th International Conference on Machine Learning. San Francisco, USA, 1997: 107-115