软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software, [doi: 10.13328/j.cnki.jos.000000]

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

基于代价 ELM 的 Bug 报告分类方法*

张天伦[1], 陈 荣[1], 杨 溪[1], 祝宏玉[2]

[1](大连海事大学 计算机科学与技术学院,辽宁 大连 116026)
[2](深圳大学 计算机与软件学院,广东 深圳 518060)
通讯作者: 陈荣, E-mail: rchen@dlmu.edu.cn

摘 要: 在所有的软件系统开发过程中,Bug 的存在基本是不可避免的问题.对于软件系统的开发者来讲,修复 Bug 最有利的工具就是 Bug 报告.但是,人工识别 Bug 报告会给开发人员带来新的负担.因此,自动对 Bug 报告进行分类,这是一个很有必要的工作.基于此,本文提出采用基于机器学习的方法来对 Bug 报告进行分类.具体地,本文主要解决 Bug 报告自动分类的三个问题.第一个是 Bug 报告数据集里不同类别的样本数量不平衡问题;第二个是 Bug 报告数据集里被标注的样本不充足问题;第三个是 Bug 报告数据集总体样本量不充足问题.为了解决这三个问题,我们分别引入了基于代价的有监督分类方法,基于模糊度的半监督学习方法以及样本迁移方法.通过在多个 Bug 报告数据集上进行实验,我们验证了这些方法的可行性和有效性.

关键词: 软件 Bug 报告;有监督分类方法;半监督学习方法;样本迁移方法
中图法分类号: TP311

中文引用格式: 张天伦,陈荣,杨溪,祝宏玉.基于代价 ELM 的 Bug 报告分类方法.软件学报. http://www.jos.org.cn/1000-9825/0000.htm

英文引用格式: Zhang TL, Chen R, Yang X, Zhu HY. The Classification of Bug Reports based on Cost-ELM. Ruan Jian Xue Bao/Journal of Software, 2018 (in Chinese). http://www.jos.org.cn/1000-9825/0000.htm

The Classification of Bug Reports based on Cost-ELM

ZHANG Tian-Lun[1], CHEN Rong[1], YANG Xi[1], ZHU Hong-Yu[2]

[1](College of Information Science and Technology, Dalian Maritime University, Dalian 116026, China)

[2](College of Computer Science and Software Engineering, Shenzhen University, Shenzhen 518060,Chian)

Abstract: It is unavoidable that Bug exists in the process of developing software system. Bug report is the powerful tool repairing Bugs for software developer. However, one can suffer from identifying a large number of Bug reports by using manual method. Due to the need for identifying Bug reports automatically, we propose a solution based on machine learning. Concretely, this paper focuses on three

* 基金项目: 国家自然科学基金(61672122, 61602077); 辽宁省自然科学基金(20170540097); 中央高校基本科研业务费资助项目(3132016348)

Foundation item: National Natural Science Foundation of China (61602077, 61602077); Natural Science Foundation of Liaoning Province of China under Grant (20170540097); Fundamental Research Funds for the Central Universities under Grant (3132016348)

收稿时间: 0000-00-00; 修改时间: 0000-00-00; 采用时间: 0000-00-00; jos 在线出版时间: 0000-00-00

CNKI 在线出版时间: 0000-00-00

problems in the field of Bug report classification. The frist one is that the imbalanced distribution of Bug report dataset in terms of Bug category; the second is that the insufficient labeled sample in Bug report dataset; the last is the limited training data available. In order to solve these issues, we propose three methods which are based on cost-supervised classification, semi-supervised learning and sample transfering, respectively. Several experiments on real Bug report datasets are conducted, and prove the practicability and effectiveness of the proposed methods.

Key words: software Bug reports; supervised classification method; semi-supervised learning; sample transfering

在软件系统开发领域里,Bug 修复[1]是一个十分关键的环节.近几年,随着软件工程项目的规模与复杂度的提升,在软件开发过程中,不可避免地会出现大量的 Bug,为了系统可以正常运行,修复这些 Bug 是十分必要的,同时修复这些 Bug 的任务量也是非常艰巨的.对于软件系统的开发人员而言,在修复 Bug 时,最常用的辅助工具就是 Bug 报告[2].Bug 报告是以文本的形式来描述 Bug 细节的数据,并且根据任务的不同,这些数据被标上不同的标签.但是 Bug 报告的质量良莠不齐,如果人工地辨别这些 Bug 报告的质量,无疑又为软件开发带来了沉重的工作量.所以,近几年,很多研究者研究如何将 Bug 报告进行自动分类,其中, Antoniol 等人[3]利用文本挖掘技术来对 Bug 报告进行分类,将 Bug 报告数据分成需即时处理的 Bug 和非即时处理的 Bug,他们所使用的技术是决策树、对率回归以及朴素贝叶斯等方法.Menzies 等人[4]利用规则学习技术将 Bug 报告分成严重 Bug 和非严重 Bug. Tian 等人[5]通过构建一个基于机器学习的预测框架,来预测 Bug 报告的优先级,在这个框架中,他们主要考虑到六个因素:时序,内容,作者,严重程度,产生原因以及相关的 Bug.除此之外,还有学者对 Bug 报告的质量进行分类.Feng 等人[6]通过提取 Bug 报告的特征,来对 Bug 报告进行高质量和低质量的区分,并且将预测得到的信息提交给软件开发者.为了降低 Bug 的冗余度,Runeson 等人[7]基于信息检索技术提出来一种计算 Bug 报告相似性的方法.类似的,Sun 等人[8]提出了一种多特征的信息检索模型,其目的也是为了计算 Bug 报告之间的相似度.近些年,更多的学者关注于 Bug 报告数据集的类别不平衡问题.其中,Lamkanfi 等人[9]通过人工选择的方法来补充数据集,解决数据集的类别不平衡问题,但是人工选择的方法局限性很大,同时也不适合于实际的工作.Yang 等人[10]利用四个解决不平衡问题的策略来处理 Bug 报告分类问题,即:随机欠采样(random under-sampling,RUS)、随机过采样 (random over-sampling,ROS)和动态少数类过采样(synthetic minority over-sampling,SMOTE)方法,这些方法在解决不平衡问题时,其效率明显优于人工选择的方法.

本文的关注点也是自动分类 Bug 报告.但是,我们更加关注于 Bug 报告分类中存在的一些问题.首先,在 Bug 报告数据集里,不同类别的样本的数量不同,而且,通常情况下,样本数量之间的差异会造成较大的不平衡度.如果一个数据集的不平衡度较大,那么通过这个数据集训练得到的分类模型的性能就会受到不良的影响.这主要是因为在训练过程中,不平衡的数据集会引导分类器更加侧重于对多数类样例的识别,在更严重的情况下,少数类的样例会被当做噪音点处理.

为了解决 Bug 报告中数据不平衡的问题,我们引入了基于代价的分类器训练策略.传统上的分类器的优化求解目标主要是最大化分类结果的精确度.而基于代价的分类器训练策略则是最小化分类器因错误分类而产生的代价.具体地,不同类别的样例被错误分类的代价应该不同,一些较易被分类的样例应该有较小的错误代价,而那些较难被分类的样例应该有比较大的错误分类代价,在最小化整体的错误分类代价时,就会使得分类器更加关注代价大的样例的分类情况.因为传统的分类器模型默认所有样例的分类代价是一样的,所以,基于代价的分类器训练方法是对传统方法的一个扩展.

在我们的工作里,我们选用的分类器模型是超限学习机(extreme learning machine,ELM).该方法由 Huang 等人[11]提出,因为其训练方法的高效性,目前被普遍用于模式识别的领域里[12].同时,为了解决上面提到的数据不平衡问题,我们用基于代价的分类器训练方法来训练 ELM 模型,训练得到的模型被称为代价敏感的 ELM 模型.

Bug 报告分类中,存在的第二个问题是:在数据集里,被标记的样本量不充足.传统的分类器大都是通过有监督算法进行训练的,这就要求数据集里的样本既有条件属性也要有与之对应的标签属性.可是,给样本指定正确的标签需要大量的精细的人工标注工作.通常情况下,人工给数据标注类别属性是繁重且耗时的.所以,为了解决这个问题,本文提出来一种可以自动标注标签的半监督学习方法.该方法主要利用弱分类器对未知标签的数

据进行类别标定,然后结合模糊度[13]这一指标选择不确定性小的数据来扩充原有的有标签的数据集.

在 Bug 报告分类中,第三个问题是:数据集的样本总量不充足.在训练分类器时,往往需要充足的训练数据.当训练数据不充分时,会导致训练得到的分类器出现严重的过拟合现象.过拟合现象主要表现为:分类器在训练集上的分类效果良好,可是在测试时或者在实际工作中,其分类效果很不理想.这主要是因为训练样本容量少导致分类器对数据的统计规律学习得不充分,最终使其泛化能力弱,不能对训练集以外的数据做出正确的判别.

为了解决这个问题,我们提出来一种样本迁移[16]的方法,即:用其他 Bug 数据集里的样本来补充数据量不充足的数据集的样本容量.自然地,如何选取样本来补充数据集,这是一个十分关键的问题.在我们的工作中,我们仍旧利用第二个问题里的方法.即:先在原数据集上训练得到一个弱分类器,然后用弱分类器对被迁移的样本进行分类,得到分类结果后,结合模糊度,选择模糊度小的样本来补充原始的数据集.在这个过程中,需要注意,不同的数据集的样本维度可能不同,为了统一维度,我们提出来使用受限玻尔兹曼机(restricted Boltzmann machines,RBM)[18]来对不同数据集的样本进行编码,即:将不同数据集里的样本编码成统一维度的数据.这样做使得我们的方法具有更加广泛的适用性.注意到,用来自不同分布的样例来补充训练集合,这个方法的可行性已经在[43]得到证明,并且在我们的试验中会进一步被验证.

综上所述,本文的贡献点主要有三个方面:第一,针对Bug报告中的类别不平衡问题,我们将基于代价的分类器训练方法引入到 Bug 报告分类问题里,并且使用 ELM 作为基本的分类模型;第二,针对 Bug 报告中被标记的样本数量不充足的问题,我们将基于模糊度的半监督方法引入到 Bug 报告分类问题里,通过扩充有标签的样本容量来增强分类效果;第三,针对 Bug 报告中数据集样本总量不充足的问题,我们将基于 RBM 和模糊度的样本迁移方法引入到 Bug 报告分类问题里,通过扩充训练集总样本容量来避免分类器出现过拟合的现象.

整体上,本文的结构安排是:第 1 节介绍本文涉及到的相关背景知识;第 2 节详细介绍本文提出来的方法;第 3 节展示本文的相关实验;第 4 节对本文的工作进行总结与展望.

1 背景知识

在这一章中,我们将详细介绍本文主体方法所涉及到的背景知识.这些背景知识主要分为两个部分:第一个是 ELM 的训练策略;第二个是 RBM 的学习方法.

本文主要关注的是对 Bug 报告数据集的分类问题.我们选用的是基于神经网络的分类器.神经网络是一种非线性的函数模型,该模型通过调节网络参数,可以拟合输入到输出的映射关系.近几年,神经网络在模式识别领域里取得了很多令人瞩目的成果[19,20,36],尤其在自然语言处理的工作里,基于神经网络的方法要明显优于其他方法的效果[21].自动识别 Bug 报告的类别,是一种关于自然语言的类别识别工作,所以我们选用神经网络作为分类模型.

按照训练方式划分,神经网络可以被分为两种类型:第一种是基于迭代优化的方法[22]来调整网络的参数;第二种是基于随机赋权的方法[23,24]来学习网络的参数.其中,ELM 作为一种随机赋权网络模型,在近几年广泛受到关注.ELM 主要的优点是训练方式高效快捷,并且在大多数领域里,其性能要优于或者等于迭代寻优算法训练出来的网络.接下来,我们详细介绍 ELM 的参数学习过程.

ELM 模型是一个单隐层的前馈神经网络,如图 1 所示.其中, I 是网络的输入层(由 nus 个节点组成), H 是网络的隐层(由 ncc 个节点组成), O 是网络的输出层(由 ns 个节点组成).输入层和隐层之间的权重 $\boldsymbol{\alpha}\in\mathbb{R}^{nus\times ncc}$ 是网络输入层权重,隐层和输出层之间的权重 $\boldsymbol{\beta}\in\mathbb{R}^{ncc\times ns}$ 是网络的输出层权重,为了增加网络模型的线性表达能力,在隐层加入了偏置矩阵 $\mathbf{B}=[\mathbf{b}_1,\mathbf{b}_2,\cdots\mathbf{b}_n]^T$,其中, $\mathbf{b}_i=(b_1,\cdots b_{ncc})^T$.为了提高网络的非线性表达能力,隐层的每一个单元节点需要被非线性函数作用,这里称这个函数为激活函数,一般地,该函数选择的是 $sigmoid$ 函数,即:

$$\sigma=sigmoid(x)=\frac{1}{1+e^{-x}} \tag{1}$$

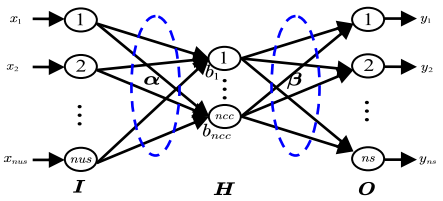


Fig.1 Feedforward neural network with single hidden layer

图 1 单隐层前馈神经网络模型

ELM 的训练方法是有监督的方法,因而在训练过程中需要有标签的数据集 **S** .在 **S** 中,条件属性矩阵用 **X** 表示,决策属性矩阵用 **T** 表示,这两个矩阵的形式如下所示:

$$\mathbf{X}=\begin{pmatrix} x_{11} & \cdots & x_{1,nus} \\ \vdots & \ddots & \vdots \\ x_{n1} & \cdots & x_{n,nus} \end{pmatrix}_{n\times nus}$$

$$\mathbf{T}=\begin{pmatrix} y_{11} & \cdots & y_{1,ns} \\ \vdots & \ddots & \vdots \\ y_{n1} & \cdots & y_{n,ns} \end{pmatrix}_{n\times ns}$$

其中, n 代表数据集 **S** 中的样本总个数; **X** 中每一行是一个 nus 维的样本向量; **T** 中每一行是一个 ns 维的标签向量,该向量用 one-hot 形式表示,且 ns 等于数据集里的类别个数。

在实际的分类工作中, I 接收输入矩阵 **X** ,经前向传播依次得到 H 的输出 $\mathbf{H}\in\mathbb{R}^{n\times ncc}$ 和 O 的输出 $\mathbf{Y}\in\mathbb{R}^{n\times ns}$:

$$\mathbf{Y}=\sigma(\mathbf{X}\boldsymbol{\alpha}+\mathbf{B})\boldsymbol{\beta}=\mathbf{H}\boldsymbol{\beta} \tag{2}$$

其中, **X** 是已知的, **α** 和 **B** 是被随机赋值的,只有 **β** 一个是未知量,也就是需要优化求解的网络参数.优化的目标是最小化网络输出 **Y** 和期望输出 **T** 之间的距离:

$$\boldsymbol{\beta}^{*}=\underset{\boldsymbol{\beta}}{argmin}(\mathbf{H}\boldsymbol{\beta}-\mathbf{T}) \tag{3}$$

β 可以用下面的等式求解:

$$\boldsymbol{\beta}=(\mathbf{H}^{T}\mathbf{H})^{-1}\mathbf{H}^{T}\mathbf{T} \tag{4}$$

很明显,通过上式求解 **β** 是一个最小二乘问题.其中,为了更一般化地定义上式的求解过程,可以求 **H** 的广义逆,即 Moore-Penrose 逆.

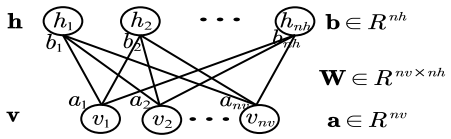


Fig.2 The structure of RBM

图 2 RBM 网络结构

接下来将介绍本文所涉及到的另一个数学模型,即:RBM.RBM 是一个概率图模型[25],其结构如图 2 所示.可以看出 RBM 是一个无向二分图模型,其中, $\mathbf{v}=(v_1,v_2\cdots v_{n_v})$ 是可见层(由 n_v 个节点构成), $\mathbf{h}=(h_1,h_2\cdots h_{n_h})$ 是隐藏层(由 n_h 个节点构成).RBM 在本文里的主要作用是将可见层的数据编码成隐藏层的表达形式.由于 RBM 的训练

机制采用的是无监督的方法,所以训练数据集 $\mathbf{S}=\{\mathbf{v}^1,\mathbf{v}^2,\cdots\mathbf{v}^N\}$ 里的样本 $\mathbf{v}^i$ 不需要标签属性, N 是样本个数.

可见层和隐藏层的条件概率公式可以由下面的公式表达:

$$\begin{cases} p(h_i=1|\mathbf{v})=\sigma(b_i+\sum_{j=1}^{n_v}w_{i,j}v_j) \\ p(v_j=1|\mathbf{h})=\sigma(a_j+\sum_{i=1}^{n_h}w_{i,j}h_i) \end{cases} \tag{5}$$

其中, σ 的函数形式是 *sigmoid* 函数.在上面的式子中, $\mathbf{a}=(a_1,a_2,\cdots a_{n_v})$ 是可见层偏置, $\mathbf{b}=(b_1,b_2,\cdots b_{n_h})$ 是隐藏层偏置, $\mathbf{W}=[w_{i,j}]_{n_v\times n_h}$ 是 RBM 的网络权重. $\mathbf{a}$, $\mathbf{b}$ 和 $\mathbf{W}$ 是需要学习的参数 θ .给定训练集合 $\mathbf{S}$,优化目标是求解下式的极大似然:

$$\ln L_{\theta,\mathbf{S}}=\ln\prod_{m=1}^{N}p(\mathbf{v}^m)=\sum_{m=1}^{N}\ln p(\mathbf{v}^m)$$

其中, $p(\mathbf{v}^m)=\frac{1}{Z_\theta}\sum_h e^{-E(\mathbf{v},\mathbf{h})}$; $E_\theta(\mathbf{v},\mathbf{h})=-\sum_{j=1}^{n_v}a_jv_j-\sum_{i=1}^{n_h}b_ih_i-\sum_{i=1}^{n_h}\sum_{j=1}^{n_v}h_iw_{i,j}v_j$.通过对上式求偏导,参数更新如下:

$$\begin{cases} \frac{\partial\ln L_{\theta,\mathbf{S}}}{\partial w_{i,j}}=\sum_{m=1}^{N}[p(h_i=1|\mathbf{v}^m)v_j^m-\sum_{\mathbf{v}}p(\mathbf{v})p(h_i=1|\mathbf{v})v_j] \\ \frac{\partial\ln L_{\theta,\mathbf{S}}}{\partial a_j}=\sum_{m=1}^{N}[v_j^m-\sum_{\mathbf{v}}p(\mathbf{v})v_j] \\ \frac{\partial\ln L_{\theta,\mathbf{S}}}{\partial b_i}=\sum_{m=1}^{N}[p(h_i=1|\mathbf{v}^m)-\sum_{\mathbf{v}}p(\mathbf{v})p(h_i=1|\mathbf{v})] \end{cases}$$

可以看出,因为 $\Sigma_{\mathbf{v}}$ 的存在,上式的计算复杂度为 $O(2^{n_v+n_h})$.为了简化计算复杂度,可以采用 Gibbs 采样[25]的方法,经过 t 次采样后,上式可以通过下面的式子近似求解:

$$\begin{cases} \frac{\partial\ln L_{\theta,\mathbf{S}}}{\partial w_{i,j}}\approx\sum_{m=1}^{N}[p(h_i=1|\mathbf{v}^{(m,0)})v_j^{(m,0)}-p(h_i=1|\mathbf{v}^{(m,t)})v_j^{(m,t)}] \\ \frac{\partial\ln L_{\theta,\mathbf{S}}}{\partial a_j}\approx\sum_{m=1}^{N}[v_j^{(m,0)}-v_j^{(m,t)}] \\ \frac{\partial\ln L_{\theta,\mathbf{S}}}{\partial b_i}\approx\sum_{m=1}^{N}[p(h_i=1|\mathbf{v}^{(m,0)})-p(h_i=1|\mathbf{v}^{(m,t)})] \end{cases} \tag{6}$$

具体地,在本文里,我们采用 Hinton 等人[27]提出来的对比散度(contrastive divergence,CD)算法来优化 RBM 模型的参数,优化过程如算法 1 所示.

算法 1. RBM 的训练算法.

(1) 输入:训练集 $\mathbf{S}$

(2) 初始化:迭代次数 *Iter* ,学习率 η ,隐层节点数 n_h ;随机初始化网络参数 $(\mathbf{W},\mathbf{a},\mathbf{b})$

(3) 令: $\mathbf{W}^1=\mathbf{W}$, $\mathbf{a}^1=\mathbf{a}$, $\mathbf{b}^1=\mathbf{b}$

(4) for $t=1$ to *Iter* do

通过公式 6 计算参数梯度 $(\nabla\mathbf{W}^t,\nabla\mathbf{a}^t,\nabla\mathbf{b}^t)$

通过下面的公式更新参数:

$$\begin{cases}\mathbf{W}^{t+1}=\mathbf{W}^{t}+\eta(\frac{1}{N}\nabla\mathbf{W}^{t})\\ \mathbf{a}^{t+1}=\mathbf{a}^{t}+\eta(\frac{1}{N}\nabla\mathbf{a}^{t})\\ \mathbf{b}^{t+1}=\mathbf{b}^{t}+\eta(\frac{1}{N}\nabla\mathbf{b}^{t})\end{cases}$$

end

(5) 令: $\mathbf{W}^{*}=\mathbf{W}^{Iter+1}$, $\mathbf{a}^{*}=\mathbf{a}^{Iter+1}$, $\mathbf{b}^{*}=\mathbf{b}^{Iter+1}$

(6) 返回 $(\mathbf{W}^{*},\mathbf{a}^{*},\mathbf{b}^{*})$

2 方法设计

在这一章中,我们将详细介绍我们提出来的方法,并且将这个新的方法用于解决 Bug 报告分类中普遍存在的三个问题:第一个问题是数据类别不平衡问题,第二个是训练集里被标记的数据不充足的问题,第三个是数据集之间的样本迁移问题.

2.1 基于代价ELM解决数据不平衡问题

ELM 是一种随机赋权网络模型,同其他神经网络一样,是一种非线性函数.在分类问题里,ELM 将条件属性域里的样本映射为决策属性域里的类别标签.在这一点上,ELM 和其他分类器一样,都是通过学习样例到标签之间的映射关系来进行分类工作.ELM 的具体学习过程在第二章里已经详细介绍,这个学习过程的优化目标就是最小化实际输出和期望输出之间的距离,这个距离通常是二者之间的欧氏距离.在分类问题中,网络的期望输出是样例的类别标签(一般将其表达成 one-hot 形式的向量),通过最小化网络的实际输出向量与期望输出向量之间的欧氏距离,来学习得到网络的参数.

可是,有一个问题需要注意,在分类问题里,每一类的样例被错误分类的代价应该是不同的,比如在不平衡数据集里,多数类样例因为样本个数多,所以更容易被正确分类,而少数类样例则更容易被错误分类,所以一个多数类样例被错误分类的代价应该小于一个少数类样例被错误分类的代价.但是在 ELM 以及大多数分类器的前提假设里,都是默认每个类别的样例被错误分类的代价是一样的.

本文主要是解决 Bug 报告的严重性分类问题.在实际工作中,严重的 Bug 需要被即时处理,不严重的 Bug 可以被延时处理.在自动分类 Bug 时,就需要准确预测 Bug 报告的严重程度,将严重的 Bug 识别出来,即时地提交开发人员处理.但是,在大多数 Bug 报告数据集里,严重的 Bug 样例个数要明显多于不严重的 Bug 样例个数,因此 Bug 报告数据集通常都是类别不平衡的数据集.这样导致分类器易将不严重 Bug 误报成严重 Bug,降低了分类器对严重 Bug 的识别准确率.为此,我们对 ELM 模型在分类问题上进行扩展,得到一种考虑到错误代价的分类模型.因此,我们引入代价敏感的 ELM 分类模型[29].

为了构建代价敏感的 ELM 分类模型,首先需要构造一个错误分类的代价矩阵 $\mathbf{D}$,即:

$$\mathbf{D}=\begin{pmatrix}d_{11} & d_{12} & \cdots & d_{1,ns}\\ d_{21} & d_{22} & \cdots & d_{2,ns}\\ \vdots & \vdots & \ddots & \vdots\\ d_{ns,1} & d_{ns,2} & \cdots & d_{ns,ns}\end{pmatrix}_{ns\times ns}$$

在这个矩阵里, $d_{i,j}$ 代表第 i 类样例被错误分到第 j 类的代价大小,很明显地, $d_{i,i}$ 等于零,也就是第 i 类样例被正确分类时所产生的代价是零.我们用 c_i 代表第 i 类样例被错误分类的代价, c_i 可以用下面的式子求得:

$$c_i=\sum_{j=1}^{ns}d_{i,j} \tag{7}$$

假设数据集里的类别个数一共是 ns,那么通过等式 7,所有类别的样例被错误分类的代价可以被集合 **Cost** 表示: $\mathbf{Cost}=(c_1,c_2,\cdots c_{ns})$.

通过引入代价矩阵,代价敏感的 ELM 将原始的 ELM 扩展成了一个考虑到错误分类代价的分类模型.在优化过程中,代价敏感的 ELM 将优化目标扩展成为一个带有代价权重的欧氏距离之和.为了表示这个优化目标,首先将训练集里的样例按照类别分组,同上,这里假设类别个数为 ns,每一组样例同属于一个类别,这个组可以被表示为 $\mathbf{X}_i$,被分组后的数据集可以被表示为 $\mathbf{X}=(\mathbf{X}_1^T,\mathbf{X}_2^T\cdots\mathbf{X}_{ns}^T)^T$,其对应的标签矩阵为 $\mathbf{T}$,对应的隐层矩阵 $\mathbf{H}$ 可以由下面的公式求出:

$$\mathbf{H}=(\mathbf{H}_1^T,\mathbf{H}_2^T\cdots\mathbf{H}_{ns}^T)^T=\sigma(\mathbf{X}\alpha+\mathbf{B}) \tag{8}$$

代价敏感的 ELM 的优化目标是最小化分类错误代价,通过上面给出的一些符号化定义,这个优化目标可以被表示为:

$$\begin{aligned}\boldsymbol{\beta}^*&=\underset{\beta}{argmin}\left\|\begin{pmatrix}c_1H_1\beta\\c_2H_2\beta\\\vdots\\c_{ns}H_{ns}\beta\end{pmatrix}-\begin{pmatrix}c_1T_1\\c_2T_2\\\vdots\\c_{ns}T_{ns}\end{pmatrix}\right\|\\&=\underset{\beta}{argmin}(c_1\|H_1\beta-T_1\|+c_2\|H_2\beta-T_2\|+\cdots+k_{ns}\|H_{ns}\beta-T_{ns}\|)\end{aligned} \tag{9}$$

这是一个加权的最小二乘解问题,令: $\mathbf{H}_c=(c_1\mathbf{H}_1^T,c_2\mathbf{H}_2^T\cdots c_{ns}\mathbf{H}_{ns}^T)^T$, $\mathbf{T}_c=(c_1\mathbf{T}_1^T,c_2\mathbf{T}_2^T\cdots c_{ns}\mathbf{T}_{ns}^T)^T$.在求解这个优化问题时,同样可以用最小二乘法来求解最优解,即:

$$\boldsymbol{\beta}^*=(\mathbf{H}_c^T\mathbf{H}_c)^{-1}\mathbf{H}_c^T\mathbf{T}_c \tag{10}$$

同第一章里介绍的 ELM 参数优化求解方法一样,这里的矩阵求逆可以通过 Moore-Penrose 广义逆来求解得到.

在本文所要解决的 Bug 报告分类问题中,数据集都是二分类问题,其中多数类是严重的 Bug 报告,少数类是不严重的 Bug 报告,通常这两类之间会存在很大的不平衡度[9,10],因此,我们在这里提出来用代价敏感的 ELM 来分类 Bug 报告数据集里的样例,通过给少数类样例比较大的错误分类代价,给多数类样例比较小的错误分类代价,并且最小化错误分类的总代价,来自动地解决 Bug 报告数据集在分类过程里存在的类别不平衡问题.

其中,所有数据集上的代价矩阵是一个 2×2 的方阵,对角线上元素全为零,代表着样例被正确分类的代价是零,其它元素则分别代表着少数类样例被错分类到多数类的错误代价 c_a 和多数类样例被错分到少数类的错误代价 c_b,在设置代价矩阵时,c_a 要大于 c_b,之后通过最小化错误分类的总代价,可以使得 ELM 模型对少数类的错误分类更加敏感,从而达到解决分类中的类别不平衡问题.用代价敏感的 ELM 来解决 Bug 报告分类中不平衡问题的过程可以用算法 2 来表示.

算法 2. 代价敏感的 ELM 的训练算法.

(1) 输入:条件属性矩阵 $\mathbf{X}$,标签属性矩阵 $\mathbf{T}$ //输入的矩阵已按类别分组

(2) 初始化:隐层节点数 ncc,代价矩阵 $\mathbf{D}$;随机初始化:输入权重 $\boldsymbol{\alpha}$,隐层偏置 $\mathbf{B}$

(3) 用公式 8 计算隐层输出值矩阵 $\mathbf{H}$

(4) 通过公式 10 求解 $\boldsymbol{\beta}^*$

(5) 返回 $\boldsymbol{\beta}^*$

2.2 基于半监督方法处理训练数据不充分问题

本文之前介绍的 ELM 方法和代价敏感的 ELM 方法,都是有监督学习的方法.监督学习的方法需要训练数据既有条件属性也要有与之相对应的标签属性.但是,在软件测试的过程中,条件属性容易获得,与之对应的标

签属性则需要精密的人工标注才可以获得.面对这种情况,我们在本节提出来一种基于模糊度的半监督学习方法[29],并且用这种方法来解决软件 Bug 报告的分类问题.在[29]里,已经证明,高模糊性的样例的分类代价要明显大于低模糊性的样例的分类代价,所以这里使用低模糊性的样例来填充原数据集可以避免整体的分类代价过高,而且 Wang 等人[13]提出,模糊性与不确定性存在着正相关的关系,低模糊性的样例具有低的不确定性,也就是被正确分类的可能性较大,这也是我们选择低模糊性样例来填充数据集的原因.

如上所述,在软件测试过程中,会产生很多无标签的数据,而软件 Bug 报告分类问题主要就是通过学习条件属性到标签数据之间的映射关系,使得分类器可以自动分类这些数据.目前,在机器学习领域,都是使用数据驱动的分类方法,这就需要在训练这些分类器时,可以提供大量的有标签的训练数据.本文里这些数据的标签只有严重和非严重两类.但是要想获取这些标签,就需要大量的人工标注,人工标注的工作往往耗时而且代价昂贵,这就很自然地使人们想到用未标注的数据来解决软件测试 Bug 报告的分类问题.

在利用未标注的软件测试 Bug 报告数据时,我们提出了一个方法,即:通过在有限的有标注的数据集上训练出一个弱分类器,然后用这个弱分类器对未标注的数据进行分类,得到分类结果后,我们通过模糊度来衡量这些分类结果的可信程度,选择可信程度高的数据来补充原始的训练数据集,而这些数据的标签属性则是通过弱分类器给出.

理论上,上述过程可以重复多次,每一次都从未标注的数据集里挑选出可信程度大的数据来补充训练数据集.因为训练数据集的样本容量在不断地扩充,所以训练出来的分类器的分类性能也会逐渐提高.最终的理想结果是,未标注的样例全部被自动地加上标签并且被扩充到训练集里,通过这个完备的数据集,可以训练得到一个泛化能力很强的分类器.但是需要注意,当分类器的分类精度不是很高时,错误分类的情况很可能会发生,也就是未标注的样例在这种情况下很可能被标上错误的标签,这样就会干扰分类器性能的提升.因而,要想达到之前所述的理想的情况,每次选择多少未标注的样本来扩充训练集,这是一个很关键的问题.

在实现这个方法的过程中,最关键的指标是未标注数据的模糊度的测量.模糊度的计算有多种方法,Zadeh 等人[34]总结出了一些计算模糊度的方法,其中包括:基于 Hamming 距离或者欧式距离的模糊性的计算方法,针对大型多类问题的模糊性计算方法.在本文中,我们利用下面的方法来计算模糊度的大小:

$$\mathcal{F}(\boldsymbol{\mu}_i)=-\frac{1}{ns}\sum_{j=1}^{ns}(\mu_{ij}\log\mu_{ij}+(1-\mu_{ij})\log(1-\mu_{ij})) \tag{11}$$

其中,向量 $\boldsymbol{\mu}_i=(\mu_{i1},\mu_{i2}\cdots\mu_{i,ns})$, μ_{ij} 是第 i 个样本属于第 j 类的隶属度大小.

因为考虑到了数据集里类别不平衡的因素,所以在实现本节的方法时,分类器采用的是代价敏感的 ELM 模型.但是注意到,在计算模糊度时,需要知道样例属于每一个类别的后验概率,可是在 ELM 的原始设计中,网络输出的值 $\mathbf{Y}=[y_{ij}]_{n\times ns}$ 不能表达这个信息.基于此,我们引入了 *softmax*,将该函数作为激活函数作用到网络的输出层,计算公式如下所示:

$$\mu_{ij}=\frac{e^{y_{ij}}}{\sum_{k=1}^{ns}e^{y_{ik}}}$$

容易验证, $\mu_{ij}>0,\Sigma_{j=1}^{ns}\mu_{ij}=1$.对于用 *softmax* 的输出来表示后验概率的合理性说明，请参考附录中的相关内容.

算法 3 详细表达了上述的基于模糊度的半监督方法.

算法 3. 半监督训练算法.

(1) 输入:有标签的条件属性矩阵 **X**,标签属性矩阵 **T**;无标签的条件属性矩阵 **Z**

(2) 初始化:隐层节点个数 *ncc*,代价矩阵 **D**;随机初始化 ELM 的输入层权重和隐层偏置

(3) 根据算法 2,利用数据 **X** 和 **T** 训练一个代价敏感的 ELM: *CELM*

(4) 利用 *CELM* 对样本矩阵 **Z** 进行分类

(5) 得到 **Z** 上的分类结果后,通过公式 11 计算每个样本的模糊度

(6) 对模糊度进行升序排列,选择前 k 个样例 z_x 补充数据集 $\mathbf{X}$,

用 z_x 被 *CELM* 分类得到的标签 z_y 补充 $\mathbf{T}$

(7) 在补充后的数据集上重新根据算法 2 训练代价敏感的分类器

(8) 返回新得到的分类器的参数矩阵 $\boldsymbol{\beta}$

2.3 Bug报告数据集间的样例迁移方法

本章第二节里介绍了一种基于模糊度的半监督方法,这个方法主要为了解决训练数据中标注样本少的问题.这个方法的提出是基于一个假设,即未标注的样本和少量的有标注的样本来自同一个数据集.但是在软件测试 Bug 报告的数据集里,不仅被标注的样本量少,而且整体的样本总量也很少.在训练分类器时,训练数据集的样本量不充足会带来的问题是使得训练得到的分类器容易过拟合,也就是该分类器在训练集上表现良好,可是其泛化能力很弱,在实际工作中,泛化能力弱的分类器在处理新的待分类样本时,往往会给出错误的决策.

为了解决 Bug 报告中有些数据集的样本总量过少的问题,我们在本节提出来一种基于模糊度的样例迁移方法.该方法不需要被迁移的样例与已有样例来自同一数据集,即:为了扩充训练集的样本容量,可以使用该方法将别的数据集里的样例迁移到训练集里来扩充样本容量.

该方法的好处就是可以解决因为样本容量少而导致的过拟合问题.但是,在选择样例上,该方法存在着同上一节一样的问题,如果样本选择不好,则同样会导致过拟合的问题出现.为了更加客观的选择样本,我们先在已有的数据集上训练得到一个弱分类器,然后用该分类器对被迁移数据集进行分类,得到分类结果后,可以计算每个样例的模糊度,模糊度越大,不确定性越大,所以选择模糊度小的样例来补充原有的数据集,这些样例的标签通过弱分类器得到.然后在扩展后的数据集上再次训练分类器.这个过程可以一直重复,直到得到合适的训练样本容量.

需要注意的是,为了避免数据集里类别不平衡问题的干扰,在这个方法里,我们选择的分类器同样是代价敏感的 ELM 模型.在实际工作中,另一个需要解决的问题是,不同数据集的样本维度通常不一样,这个问题限制了这个方法的应用范围.为了解决这个问题,使得我们的方法更具一般性,我们将 RBM 引入到该方法中.

在第一章里已经介绍了 RBM.RBM 可以被视为一种样本编码工具,即可以通过非线性变换,将样本从一个维度空间映射到另一个维度空间.在编码的过程中,RBM 保证编码前和编码后的样本是等价的.在本节的工作里,我们采用 RBM 将不同数据集里的样本编码成统一的维度,这样就为分类器在不同数据集上进行迁移训练提供了可能性.我们用 $\mathbf{S}=\{\mathbf{X},\mathbf{T}\}$ 表示原数据集;同时,用 $\mathbf{S}_t$ 表示被迁移数据集 $\mathbf{S}_t=\{\mathbf{X}_t\}$.基于这些符号化的定义,该节提出来的方法可以用算法 4 来详细表述.

算法 4. 样本迁移算法.

(1) 输入:数据集 $\mathbf{S}$,其中,条件属性矩阵 $\mathbf{X}$,标签属性矩阵 $\mathbf{T}$,被迁移的数据集 $\mathbf{S}_t=\{\mathbf{X}_t\}$

(2) 初始化:ELM 的隐层节点个数 ncc,代价矩阵 $\mathbf{D}$,RBM 的隐层节点个数 n_h

随机初始化:ELM 的输入层权重和隐层偏置

(3) 通过算法 1,利用数据 $\mathbf{X}$ 和 $\mathbf{X}_t$ 分别训练得到两个 RBM 模型

(4) 通过训练得到的 RBM 对 $\mathbf{X}$ 和 $\mathbf{X}_t$ 进行编码,得到新的数据表达形式 $\mathbf{X}_p$ 和 $\mathbf{X}_{tp}$

(5) 根据算法 2,利用数据 $\mathbf{X}_p$ 和 $\mathbf{T}$,训练得到一个代价敏感的 ELM: *CELM*

(6) 利用 *CELM* 对数据集 $\mathbf{X}_{tp}$ 里的样本进行分类

(7) 得到 $\mathbf{X}_{tp}$ 上的分类结果后,通过公式 11 计算每个样本的模糊度

(8) 对模糊度进行升序排列,选择前 k 个样例 tr_x 补充数据集 $\mathbf{X}_p$

用 tr_x 被 *CELM* 分类得到的标签 tr_y 补充 $\mathbf{T}$

(9) 在补充后的数据集上重新根据算法 2 训练代价敏感的分类器

(10) 返回新得到的分类器的参数矩阵 **β**

3 本文实验

本文针对软件测试 Bug 报告数据集里存在的三个问题提出了三个解决方法,即:针对训练数据集类别不平衡问题,提出了代价敏感的 ELM 分类方法;针对数据集被标记的样本较少的问题,提出了基于模糊度的半监督方法;针对训练数据集总体样本量较少的问题,提出了基于模糊度和 RBM 的样本迁移方法.为了验证我们提出的方法的有效性,我们开展了三组验证性的实验.

在本章中,实验所采用的数据集是真实的软件测试 Bug 数据集.数据集来自三个开源项目.本章实验里用到的数据集一共 21 个,其中 7 个 Eclipse[39],7 个 Moizlla [40],7 个 GNOME[41].数据集的名称以及数据集的不平衡度都在表 1-3 里被列出.其中,不平衡度 Imbalance Ratio (IR)的计算如下所示:

$$IR = num\_maj / num\_min$$

其中, num_min 是少数类的样本个数, num_maj 是多数类的样本个数.

Table 1 Datasets of Eclipse Bug reports

表 1 Bug 报告数据集

项目	名称		IR
Eclipse	1	CDT_cdt-core	4.0175
	2	JDT_Core	2.5684
	3	JDT_Debug	2.4261
	4	PDE_UI	2.6633
	5	Platform_Debug	1.7414
	6	Platform_SWT	4.9232
	7	Platform_UI	2.5422

Table 2 Datasets of GNOME Bug reports

表 2 GNOME Bug 报告数据集

项目	名称		IR
GNOME	1	ekiga_general	9.5
	2	Evolution_Calendar	4.6262
	3	Evolution_Contacts	2.7764
	4	Evolution_Mailer	2.902
	5	Evolution_Shell	2.4444
	6	gnome-panel_Panel	3.9424
	7	gnome-terminal_general	12.6742

Table 3 Datasets of Moizlla Bug reports

表 3 Moizlla Bug 报告数据集

项目	名称		IR
Moizlla	1	Core_Printing	7.018
	2	Core_XPCOM	2.0201
	3	Core_XPConnect	5.3
	4	Core_XUL	4.0902
	5	Thunderbird_General	4.0511
	6	Core_Layout	2.8615
	7	Core_Security_UI	6.2672

在接下来的实验里,我们采用的评价指标分别是准确率和加权的 *F-measure* [42].其中,准确率是样本被正确

分类的比率,加权的 *F-measure* 是两类样例在查全率和查准率上的一个综合性的指标,在本文实验里,该指标更能反映出分类器对严重 Bug 的误报程度.准确率的计算如下所示:

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

其中, *Accuracy* 是准确率, TP, FP, TN 和 FN 的关系可以用表 4 表示.

F-measure 的计算如下所示:

$$F\text{-}meausre = \frac{(1+\gamma^2) \times Precision \times Recall}{\gamma^2 \times Precision + Recall}$$

其中,在本文实验里, γ 等于 1, *Precision* 代表查准率, *Recall* 代表查全率,这两个指标的计算可由下面的公式表达:

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TN}{TN + FP}$$

本文所用到的加权 *F-measure* 计算如下:

$$wF = W \bullet F - measure$$

其中, wF 是加权的 *F-measure*, $W = (w_1, w_2)$, w_1是第一类样例占的比例, w_2 是第二类样例占的比例.

Table 4 Confusion Matrix

表 4 混淆矩阵

Confusion Matrix		Actual Severity	
		non-severe	severe
Predicted Severity	non-severe	TP: true positives	FP: false positives
	severe	FN: false negatives	TN: true negatives

本章的第一个实验,是验证代价敏感的 ELM 在不平衡数据集上的分类效果.作为对比,我们用传统的 ELM 在相同的数据集上进行实验.除此之外,我们还比较了基本的分类器,即朴素贝叶斯模型(NB),BP 神经网络模型(BPNN)和支持向量机(SVM).数据集的划分是:把数据集平均分成 5 份,每次随机取 3 份作为训练集,剩下的 2 份作为测试集.为了保证客观性,我们对两个方法都进行了 100 次实验,最后的结果取这 100 次实验的平均值.在每次实验里,为了保证公平性,代价敏感的 ELM 和传统的 ELM 的隐层节点数,以及 BPNN 的隐层节点数都被设置成一样的个数,并且两个模型的输入层权重和隐层偏置也都被相同的随机数赋值.我们在实验里给权重和偏置随机赋权的方法是在区间[0,1]里随机取值.表 5-10 给出了这组实验的对比结果.

Table 5 Accuracies on Eclipse

表 5 Eclipse 数据集上的测试准确率

ACC	NB	BPNN	SVM	ELM	C-ELM
Eclipse_1	0.5904	0.5794	0.4549	0.6768	**0.7286**
Eclipse_2	0.5178	0.5492	0.5519	0.5817	**0.5949**
Eclipse_3	0.5575	0.57	0.5625	0.591	**0.6069**
Eclipse_4	0.5835	0.5881	0.5263	0.6334	**0.6516**
Eclipse_5	0.5039	0.5930	0.5543	0.6054	**0.6069**
Eclipse_6	0.6097	0.6178	0.5733	0.6584	**0.6742**
Eclipse_7	0.4967	0.5922	0.5550	0.6010	**0.6118**

Table 6 Weighted *F*-measures on Eclipse

表 6 Eclipse 数据集上的加权 *F* 值

WF	NB	BPNN	SVM	ELM	C-ELM
Eclipse_1	0.6123	0.6150	0.5016	0.6481	**0.6764**
Eclipse_2	0.5281	0.5687	0.5699	0.5561	**0.5707**
Eclipse_3	0.5768	0.5666	**0.5789**	0.5595	0.5758
Eclipse_4	0.5328	0.5757	0.5498	0.5807	**0.5961**
Eclipse_5	0.4546	0.5310	0.5169	0.5294	**0.5313**
Eclipse_6	0.6578	0.6603	0.6265	0.6591	**0.6732**
Eclipse_7	0.5042	0.5767	0.5763	0.5662	**0.5786**

Table 7 Accuracies on GNOME

表 7 GNOME 数据集上的测试准确率

ACC	NB	BPNN	SVM	ELM	C-ELM
GNOME_1	0.6423	0.7854	0.7141	0.8376	**0.8583**
GNOME_2	0.4688	0.6865	0.5234	0.7810	**0.7829**
GNOME_3	0.4207	0.6070	0.5783	0.6741	**0.6791**
GNOME_4	0.5613	0.6063	0.5490	**0.7266**	0.7252
GNOME_5	0.4677	0.6114	0.5718	0.7281	**0.7303**
GNOME_6	0.4594	0.6570	0.6101	0.7616	**0.7705**
GNOME_7	0.4755	0.7146	0.7091	0.7630	**0.7800**

Table 8 Weighted *F*-measures on GNOME

表 8 GNOME 数据集上的加权 *F* 值

WF	NB	BPNN	SVM	ELM	C-ELM
GNOME_1	0.7158	0.8108	0.6707	0.8041	**0.8179**
GNOME_2	0.5210	0.7018	0.5740	0.6970	**0.7026**
GNOME_3	0.4306	0.5913	0.6014	0.5956	**0.6028**
GNOME_4	0.5843	0.6071	0.5745	0.6105	**0.6138**
GNOME_5	0.4661	0.5440	0.5809	0.5854	**0.5937**
GNOME_6	0.4990	0.6745	0.6755	0.6697	**0.6793**
GNOME_7	0.5851	0.7759	0.7778	0.7832	**0.7951**

Table 9 Accuracies on Moizlla

表 9 Moizlla 数据集上的测试准确率

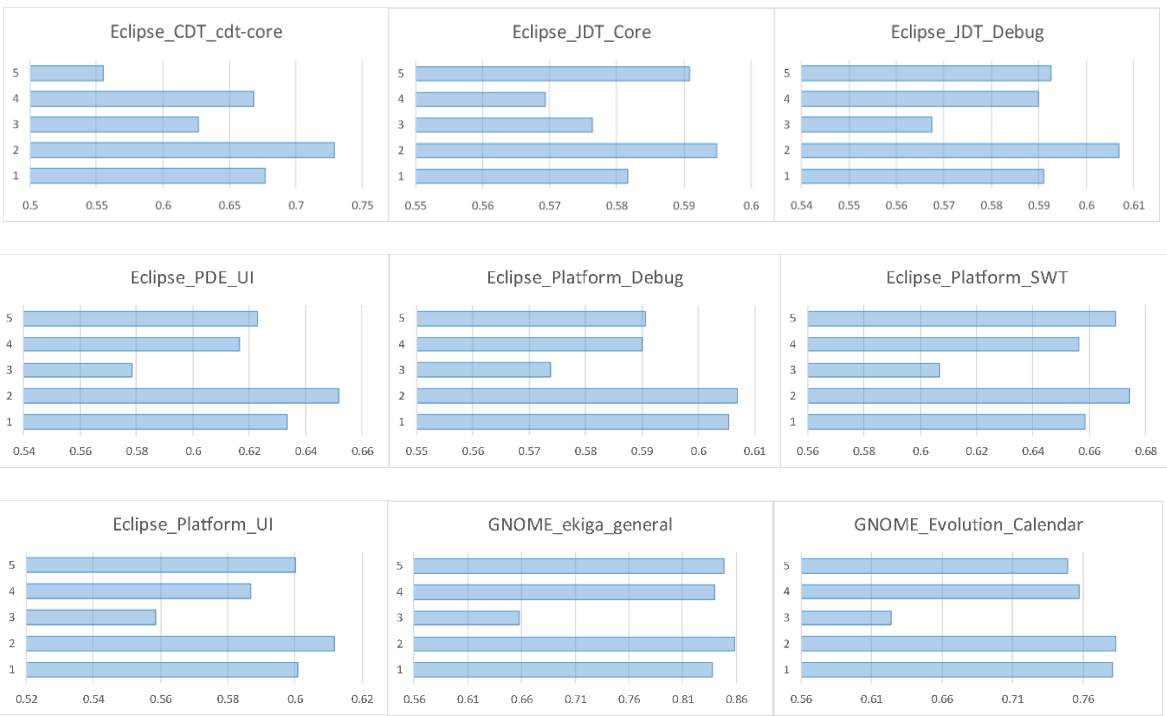
ACC	NB	BPNN	SVM	ELM	C-ELM
Moizlla_1	0.7305	0.6784	0.5613	0.7512	**0.7720**
Moizlla_2	0.4683	0.5950	0.5758	0.5922	**0.6224**
Moizlla_3	0.6667	0.6667	0.6471	0.7140	**0.7271**
Moizlla_4	0.6825	0.7143	0.6825	0.7406	**0.7821**
Moizlla_5	0.5582	0.5519	0.5440	0.5802	**0.6047**
Moizlla_6	0.5896	0.5809	0.6132	0.6356	**0.6459**
Moizlla_7	0.5634	0.6254	0.6077	0.6966	**0.7269**

Table 10 Weighted *F*-measures on Moizlla

表 10 Moizlla 数据集上的加权 *F* 值

WF	NB	BPNN	SVM	ELM	C-ELM
Moizlla_1	0.7588	0.7228	0.6341	0.7497	**0.7623**
Moizlla_2	0.5160	0.6429	0.6277	0.6222	**0.6449**
Moizlla_3	0.7119	0.6870	0.6917	0.7011	**0.7133**
Moizlla_4	0.6993	0.7019	**0.7090**	0.6835	0.7071
Moizlla_5	0.6003	0.5935	0.5897	0.5984	**0.6163**
Moizlla_6	0.6144	0.5809	**0.6369**	0.5936	0.6032
Moizlla_7	0.6263	0.6824	0.6676	0.7004	**0.7218**

通过表 1-3 的最后一列可以看出,Bug 报告数据集都是有着不平衡度的数据集,而且有些数据集的不平衡度较大.在处理数据类别不平衡这个问题时,通过给少数类样例更大的错误分类代价,得到的分类准确率在绝大多数 Bug 报告数据集上有了很明显的提升.虽然在表 7 中,GNOME_Evolution_Mailer 的准确率结果要偏低,但是通过加权的 *F-measure* 值的比较可以看出,代价敏感的分类器在重要数据样本的分类上面表现仍旧良好,也就是对严重 Bug 的误报率较低.而且通过在所有数据集上比较加权的 *F-measure* 值,可以看到,基于代价的分类模型都能取得很好的效果.与其它传统的分类器相比,ELM 方法需要设定的参数更少,尤其与 BPNN 的比较中,ELM 的训练时间要远远低于 BP 迭代调参所需要的时间,并且在绝大多数数据集上,ELM 方法可以取得较好的效果.在图 3 中,我们比较了欠采样方法(RUS-ELM),过采样方法(ROS-ELM),合成过采样方法(SMOTE)与代价敏感方法.欠采样方法是对多数类样例进行随机删减,可以看出,因为减少了训练样本的容量,并且在删减样本的过程中容易丢失重要数据,所以该方法效果不是很好;过采样的方法则是对少数类样例进行扩充,该方法可能会受到冗余数据的影响,造成分类器过拟合,对分类器泛化能力造成不好的影响;SMOTE 方法相对表现较好,但是该方法容易造成类间的 overlapping 现象,即在类别临界面生成样例时,容易将合成的多数类样例标记成少数类类别,从而不利于分类器对数据分布特点的学习.通过比较,基于代价敏感的平衡方法在绝大部分数据集上都有较好的结果,而且基于代价的方法表现得更加稳定,更适合被用于实际的工作中.从效率上来看,基于代价的方法不需要事先对数据集进行平衡化的预处理,而是可以进行端到端的训练,从而更加满足本文研究领域的需求.




GNOME_Evolution_Contacts
GNOME_Evolution_Mailer
GNOME_Evolution_Shell
GNOME_gnome-panel_Panel
GNOME_gnome-terminal_general
Moizlla_Core_Printing
Moizlla_Core_XPCOM
Moizlla_Core_XPConnect
Moizlla_Core_XUL
Moizlla_Thunderbird_General
Mozilla_Core_Layout
Mozilla_Core_Security_UI
Eclipse_CDT_cdt-core
Eclipse_JDT_Core
Eclipse_JDT_Debug
Eclipse_PDE_UI
Eclipse_Platform_Debug
Eclipse_Platform_SWT

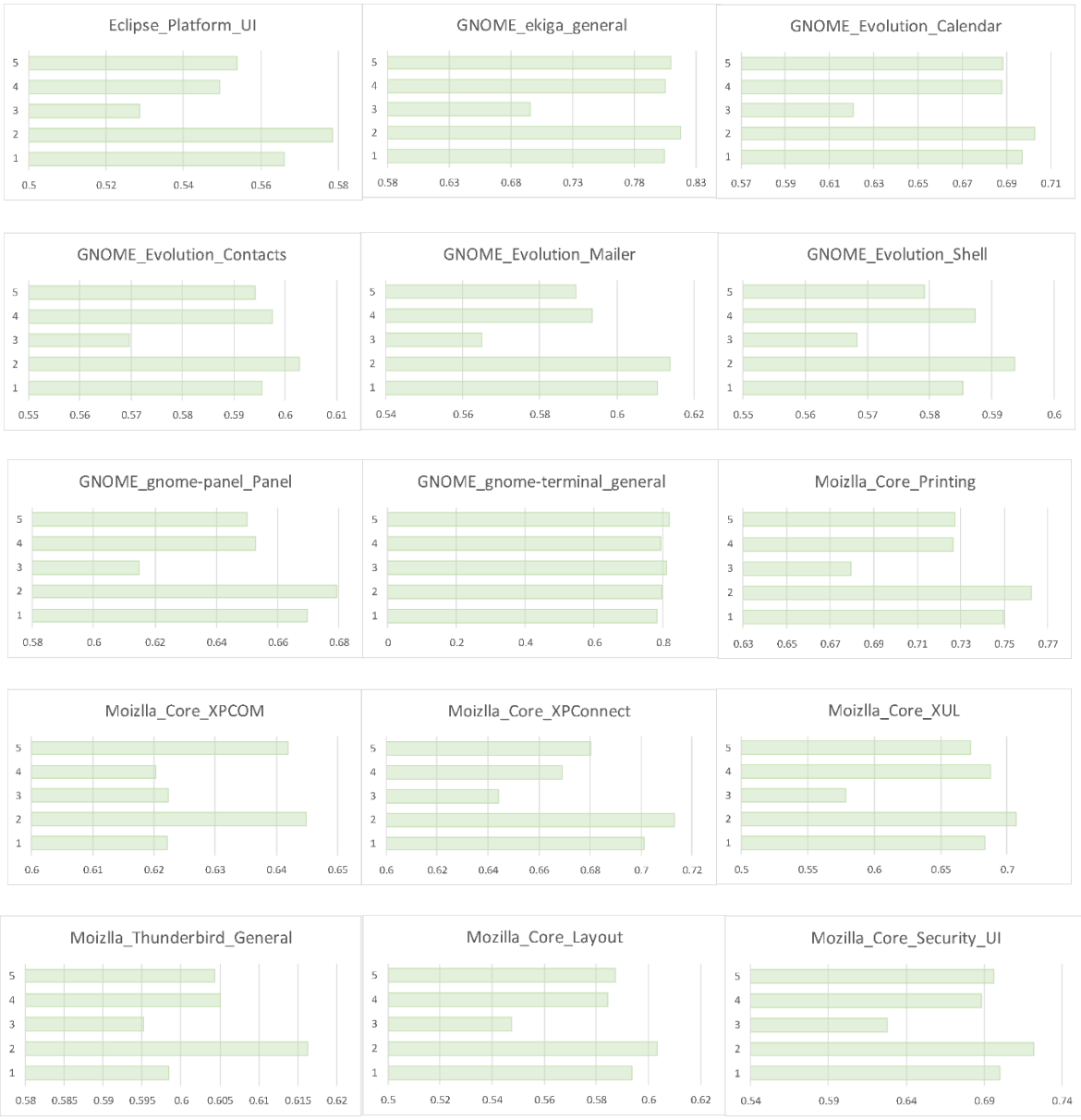


Fig.3 Comparison among RUS,ROS,SMOTE and Cost based method

图 3 欠采样,过采样,合成过采样和代价敏感方法的比较

上图中的结果都是试验重复 100 次得到的平均值,蓝色图是准确度的结果,绿色图是加权 F 值的结果.1,2,3,4,5 分别代表 ELM,CELM,RUS-ELM,ROS-ELM 和 SMOTE-ELM.

本章的第二个实验,是验证基于模糊度的半监督方法在扩充训练集样本容量上的有效性.在这个实验里,我们采用的分类器是代价敏感的 ELM 模型,目的是为了避免数据集里不平衡问题的干扰.在该实验里,对原始数据集的划分是:将数据集划分成 6 份,每次随机选择其中的 3 份作为训练集,在剩下的数据中,随机选择 2 份作为测试集,最后留下的数据作为验证集.其中,验证集则是本文第二章第二节里提到的未标注数据集.在实验过程中,我们先在原始的训练集上训练得到一个弱分类器,然后在验证集上对样例进行分类,之后通过模糊度的大小选择一定数量的验证集样例来扩充原始的训练集,然后用扩充得到的数据集再次训练一个分类器.为了保证客观

性,前后两个分类器的各项参数的设置完全一样,具体操作方法如第一个实验所述.为了通过对比显示我们方法的有效性,我们用两次训练得到的分类器对测试集的样本进行分类,最后通过上述的两个指标来评价分类结果,该结果同样是对100次的实验结果求平均值得到的最终结果.我们用first表示在原始数据集上训练得到的分类器的测试结果,用second表示用扩充后的数据集训练的分类器的测试结果.这部分的实验如表11所示.

Table 11 Results of Semi-supervised testing

表11 半监督方法的实验结果

数据集名称	准确度		wflscore	
	first	second	first	second
Eclipse_CDT_cdt-core	0.6824	**0.6997**	0.6941	**0.7037**
Eclipse_JDT_Core	0.6367	**0.6417**	0.6492	**0.6535**
Eclipse_JDT_Debug	0.6382	**0.6392**	0.6414	**0.6422**
Eclipse_PDE_UI	0.6417	**0.6424**	0.6508	**0.6509**
Eclipse_Platform_Debug	0.5964	**0.5995**	0.5971	**0.5995**
Eclipse_Platform_SWT	0.7183	**0.7312**	0.7361	**0.7446**
Eclipse_Platform_UI	0.6399	**0.6401**	0.6497	**0.6499**
GNOME_ekiga_general	0.8197	**0.8492**	0.8388	**0.8577**
GNOME_Evolution_Calendar	0.7593	**0.7732**	0.7682	**0.7789**
GNOME_Evolution_Contacts	0.7239	**0.7274**	0.7217	**0.7250**
GNOME_Evolution_Mailer	0.7222	**0.7336**	0.7253	**0.7361**
GNOME_Evolution_Shell	0.7537	**0.7559**	0.7531	**0.7551**
GNOME_gnome-panel_Panel	0.7949	**0.7975**	0.7939	**0.7959**
GNOME_gnome-terminal_general	0.9237	**0.9264**	0.9186	**0.9209**
Moizlla_Core_Printing	0.7301	**0.7605**	0.7619	**0.7811**
Moizlla_Core_XPCOM	0.7528	**0.7557**	0.7567	**0.7586**
Moizlla_Core_XPConnect	0.7061	**0.7318**	0.7322	**0.7500**
Moizlla_Core_XUL	0.7043	**0.7199**	0.7118	**0.7214**
Moizlla_Thunderbird_General	0.6988	**0.7111**	0.7095	**0.7177**
Mozilla_Core_Layout	0.6103	**0.6269**	0.6270	**0.6405**
Mozilla_Core_Security_UI	0.7785	**0.7869**	0.7841	**0.7889**

从表11可以看出,通过基于模糊度的半监督方法来扩充数据集,并且用新的数据集再次训练分类器,所得到的分类器的分类效果在每个数据集上都有了提升.这证明了我们提出来的这个方法对于改善分类器的泛化能力是有效的.

本章的第三个实验,是验证基于模糊度和 RBM 的样本迁移方法的有效性.在这个实验里,我们采用的分类器同样是代价敏感的ELM模型.我们对原始数据集的划分方法同第一个实验里数据集划分方法一样.对于被迁移的数据集我们没有采用任何的划分方法.在进行分类之前,我们首先对原始数据集的整体进行 RBM 编码,同时对被迁移的数据集的整体进行RBM编码.在这两次编码中,我们采用三个级联的RBM对输入数据进行维度空间的变换.在编码过程中,我们只需要保证两次编码的输出维度一样即可,中间隐层的维度则可以自主调节.分类实验的具体操作过程是,先在原始的训练集上训练得到一个分类器,然后对被迁移的样本进行分类,用分类结果计算每个样本的模糊度大小,根据模糊度大小来选择样本来扩充原始的训练集.之后再利用扩充后的训练集重新训练一个分类器.在实验过程中,前后两个分类器的参数设置完全一样.为了体现实验的客观性,我们重复本实验的全部过程(编码过程和分类过程)100次,最后取这100次的平均值作为实验结果,我们用before表示在原始数据集上训练得到的分类器的测试结果,用 after 表示在被扩充后的数据集上训练得到的分类器的测试结果.这些结果展示在表12中.

Table 12 Results of Transfer testing

表 12 迁移方法的试验结果

数据集名称	准确度		wflscore	
	before	after	before	after
Eclipse_CDT_cdt-core	0.7288	**0.7373**	0.7186	**0.7228**
Eclipse_JDT_Core	0.6172	**0.6187**	0.6283	**0.6295**
Eclipse_JDT_Debug	0.6146	**0.6165**	0.6223	**0.6244**
Eclipse_PDE_UI	0.6508	**0.6530**	0.6573	**0.6595**
Eclipse_Platform_Debug	0.5909	**0.5943**	0.5940	**0.5975**
Eclipse_Platform_SWT	0.7763	**0.7792**	0.7558	**0.7568**
Eclipse_Platform_UI	0.5900	**0.5911**	0.6035	**0.6040**
GNOME_ekiga_general	0.9081	**0.9117**	0.9041	**0.9092**
GNOME_Evolution_Calendar	0.8038	**0.8043**	0.8057	**0.8064**
GNOME_Evolution_Contacts	0.7289	**0.7295**	0.7324	**0.7333**
GNOME_Evolution_Mailer	0.6757	**0.6760**	0.6936	**0.6938**
GNOME_Evolution_Shell	0.7558	**0.7568**	0.7562	**0.7575**
GNOME_gnome-panel_Panel	0.7794	**0.7806**	0.7844	**0.7857**
GNOME_gnome-terminal_general	0.9264	**0.9274**	0.9199	**0.9202**
Moizlla_Core_Printing	0.8477	**0.8526**	0.8219	**0.8233**
Moizlla_Core_XPCOM	0.7822	**0.7886**	0.7677	**0.7698**
Moizlla_Core_XPConnect	0.7769	**0.7836**	0.7734	**0.7759**
Moizlla_Core_XUL	0.7429	**0.7467**	0.7404	**0.7428**
Moizlla_Thunderbird_General	0.7556	**0.7609**	0.7298	**0.7306**
Mozilla_Core_Layout	0.6609	**0.6658**	0.6626	**0.6642**
Mozilla_Core_Security_UI	0.8235	**0.8334**	0.8003	**0.8033**

从表 12 中我们可以看出,无论是准确率还是加权的 *F-measure* 值,再次训练的分类器都取得了较好的效果.这个结果显示,基于 RBM 和模糊度的样本迁移方法在扩充样本容量并且提升分类器的泛化能力这些方面是有效的.

本章的实验基本验证了我们提出来的方法的可行性和有效性.其中,如第三章所述,在利用半监督方法扩充训练集时,理论上可以进行多次迭代,逐渐扩充训练集的样本容量.在之前的实验里,我们设置的迭代次数为两次,即对分类器进行再训练的优化策略.再训练的结果已经表明我们的方法是有效的.为了进一步验证我们的方法在数据集样本量继续扩充的条件下仍旧是有效的,我们将迭代次数增加,图 4 显示了随着迭代次数增加,我们的方法在每个数据集上的表现.

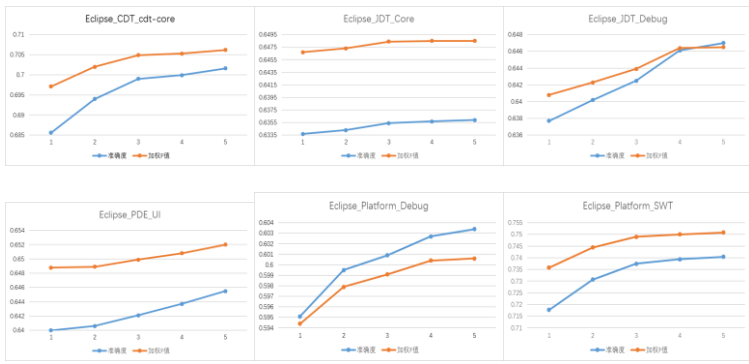


Fig.4 Results of five tests on 21 datasets

图 4 21 个数据集 5 次测试的结果

在上面的实验中,为了保证客观性,我们将实验重复了 100 次,然后取平均值.在每一次的实验中,我们提前将数据集按照 3:2:1 的比例分别分成训练集,测试集和验证集.其中验证集是用来扩充训练集样本容量的数据集合.然后,我们利用半监督方法,逐步地对训练数据集扩充了 5 次,每一次扩充后都在测试集上进行测试,并给出上图所示的测试结果.值得注意,为了克服随机赋权的影响,5 次迭代中,分类器的随机参数都是相同的值,实验结果就只会受到数据集样本容量变化的影响.从图 4 可以看出,在利用本文的半监督方法的前提下,随着数据集样本容量的增加,分类器的性能也在逐步提升,这一现象在 21 个数据集上都有体现.通过这组实验,验证了我们提出来的方法的有效性,即:在人工标记的数据不充足的条件下,可以利用有效的半监督机制,来逐步扩充训练样本集,最终可以达到提升分类性能的目的.

本章实验里网络结构参数,代价矩阵的设置以及其它参数的具体情况都详细列在了本文的附录里.

4 总结

本文主要解决软件测试 Bug 报告中的样例分类问题.在实际工作中,我们发现 Bug 报告数据集普遍存在着三个问题:第一个是样本类别不平衡问题,第二个是数据集里被标注的样本个数不充足的问题,第三个是数据集的样本总容量不充足的问题.这三个问题都会对分类工作产生不良的影响.为了解决这三个问题,我们提出来了三个方法.第一个是基于代价敏感的 ELM 模型来增加少数类样例被错误分类的代价,从而解决类别不平衡问题

带来的影响;第二个是基于模糊度的半监督方法来自动标注没有标签的数据,从而扩充有标签的数据集;第三个是基于 RBM 和模糊度的样本迁移方法来扩增原有数据集的方法.本文通过几个实验验证了我们提出来的方法是有效的,可行的.最后,可以注意到,我们提出来的方法可以被用在不同的分类器上.本文接下来的研究工作就是在不同的分类器上集成我们的方法,在 Bug 报告分类领域取得更好的效果.

References:

[1] Xia X, Lo D, Wang XY, Zhou B. Accurate developer recommendation for bug resolution. In: Proc. of the WCRE Congress. 2013. 72-81.

[2] Guo SK, Chen R, Li H. Using Knowledge Transfer and RoughSet to Predict the Severity of Android Test Reports via Text Mining. Symmetry, 2017,9(8):161.

[3] Antoniol G, Ayari K, Penta MD, Khomh F, Gueheneuc YG. Is it a bug or an enhancement?: a text based approach to classify change requests. Proceedings of the conference of the center for advanced studies on collaborative research, 2008,304-318.

[4] Menzies T, Marcus A. Automated severity assessment of software defect reports. In: Proc. of the ICSM Congress. 2008. 346-355.

[5] Tian Y, Lo D, Xia X, Sun CN. Automated prediction of bug report priority using multi-factor analysis. Empirical Software Engineering, 2015, 20(5):1354-1383.

[6] Feng Y, Chen ZY, Jones JA, Fang CR, Xu BW. Test report prioritization to assist crowdsourced testing. In: Proc. of the FSE Congress. 2015. 225-236.

[7] Runeson P, Alexandersson M, Nyolm O. Detection of duplicate defect reports using natural language processing. In: Proc. of the ICSE Congress. 2007. 499-510.

[8] Sun c, Lo D, Khoo S C. Towards more accurate retrieval of duplicare bug reports. In: Proc. of the ASE Congress. 2011. 253-262.

[9] Lamkanfi A, Demeyer S, Giger E, Goethals B. Predicting the severity of a reported bug. In: Proc. of the MSR Congress. 2010. 1-10.

[10] Yang XL, Lo D, Xia X, Huang Q, Sun JL. High-Impact Bug Report Identification with Imbalanced Learning Strategies. J. Comput. Sci. Technol, 2017,32(1):181-198.

[11] Huang GB, Zhu QY, Siew CK. Extreme learning machine: theory and applications. Neuro-computing, 2006,70(1):489–501.

[12] Huang GB, Zhou H, Ding X. Extreme learning machine for regression and multicalss classification. IEEE Trans. Syst. Man, Cybern. B, Cybern, 2012,42(2):513-529.

[13] Wang XZ, Xing HJ, Li Y, et al. A Study on Relationship between Generalization Abilities and Fuzziness of Base Classifiers in Ensemble Learning. IEEE Trans. on Fuzzy Systems, 2015,23(5):1638-1654.

[14] Qiu TY, Shen FR, Zhao JX. Review of Self-Organizing Incremental Neural Network. Ruan Jian Xue Bao/Journal of Software, 2016,27(9):2230-2247 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/5068.htm [doi: 10.13328/j.cnki.jos.005068]

[15] Wang LJ, Li M, Cai SB, Li G, Xie B, Yang FQ. Internet Information Search Based Approach to Enriching Textual Descriptions for Public Web Services. Ruan Jian Xue Bao/Journal of Software, 2012, 23(6):1335-1349 (in Chinese with English abstract). [10.3724/SP.J.1001.2012.04088]

[16] Pan JL, Tsang IW, Kwok JT, Yang Q. Domain Adaptation via Transfer Component Analysis. IEEE Trans. on Neural Networks, 2011,22(2):199-210.

[17] Furao Shen, Hui Yu, Keisuke Sakurai and Osamu Hasegawa, "An incremental online semi-supervised active learning algorithm based on a self-organizing incremental neural network", Neural Comput & Applic, 20:1061–1074, 2011 .

[18] Hinton GE. A practical guide to training restricted Boltzmann machines. Montreal: Department of Computer Science, University of Toronto, 2010.

[19] Huang GB, Zhou H, Ding X. Extreme learning machine for regression and multicalss classification. IEEE Trans. on Syst. Man, Cybern. B, Cybern, 2012,42(2):513-529.

[20] Han K, Yu D. Speech Emotion Recognition Using Deep Neural Network and Extreme Learning Machine. INTERSPEECH, 2014. 223-227.

[21] Zhang Y, Wallace BC. A Sensitivity Analysis of (and Practitioners' Guide to) Convolutional Neural Networks for Sentence Classification. https://arxiv.org/abs/1510.03820.

[22] Hinton GE. A fast learning algorithm for deep belief nets. Neural Computation, 2006,18(7):1527-1554.

[23] Wang XZ, Zhang TL, Wang R. Non-Iterative Deep Learning: Incorporating Restricted Boltzmann Machine into Multilayer Random Weight Neural Networks. IEEE Trans. on Systems, Man, and Cybernetics: Systems, 2017,99.

[24] Huang GB. An insight into extreme learning machines:Random neurons, random features and kernels. Cognit Comput, 2014,6(3): 376-390.

[25] Koller D, Friedman N. Probabilistic Graphical Models: Principles and Techniques. MIT Press, 2009.

[26] Lili Mou, Ge Li, Lu Zhang, Tao Wang, Zhi Jin, "Convolutional Neural Networks over Tree Structures for Programming Language Processing." Proceedings of 2016 AAAI Conference on Artificial Intelligence (AAAI 2016), pages 1287-1293, Phoenix, USA, January 12-18, 2016.

[27] Hinton GE. Training products of experts by minimizing contrastive divergence. Neural Computation, 2002,14(8):1771-1800.

[28] Huang GB, Li MB, Chen L. Incremental extreme learning machine with fully complex hidden nodes. Neurocomputing, 2008,71(4-6): 576-583.

[29] Zhu HY, Wang XZ. A cost-sensitive semi-supervised learning model based on uncertainty. Neurocomputing, 2017,(251):106-114.

[30] Zhao Meng, Lili Mou, Ge Li and Zhi Jin, Context-Aware Tree-Based Convolutional Neural Networks for Natural Language Inference, Proceedings of 9th International Conference on Knowledge Science, Engineering and Management (KSEM 2016), Passau, Germany, October 4-8, 2016, LNAI 9983, pp. 515–526

[31] Zhai JH. Fusion of extreme learning machine with fuzzy integal. Fuzziness and Knowledge-Based Systems, 2013,21:23-24.

[32] Huang GB, Chen L, Siew CK. Universal approximation using incremental constructive feedforward with random hidden nodes. IEEE Trans. on Neural Network,2006,17(4):879-892.

[33] Huang GB, Chen L. Enhanced random search based incremental extreme learning machine. Neurocomputing, 2008,71(16-18): 3460-3468.

[34] Zadeh LA. Fuzzy sets. Information & Control, 1965,8(3):338-353.

[35] Xiaofeng Shi,Guoqiang Xu, Furao Shen, Jinxi Zhao, "Solving the Data Imbalance Problem of P300 Detection via Random Under-Sampling Bagging SVMs", Proc. IJCNN2015, Killarney, Ireland, 2015

[36] Zeng YJ, Xu X, Fang YQ, Zhao K. Traffic Sign Recognition Using Extreme Learning Classifier with Deep Convolutional Features. Springer International Publishing, 2015.

[37] Zhai JH, Xu HY, Wang XZ. Dynamic ensemble extreme learning machine based on sample entropy. Soft Computing, 2012,16(9): 1493-1502.

[38] Huang GB, Song S, Gupta JND. Semi-supervised and unsupervised extreme learning machines. IEEE Trans on Cybern. 2014, 44(12):2405-2417.

[39] Eclipse, http://bugs.eclipse.org/bugs, 2/2/2018 available.

[40] Mozilla, http://bugzilla.mozilla.org, 2/2/2018 available.

[41] GNOME, http://bugzilla.gnome.org, 2/2/2018 available.

[42] Zhou Y, Tong Y, Gu R, Gall H. Combing Text Mining and Data Mining for Bug Report Classification. In: Proc. of the ICSME Congess. 2014. 311-320.

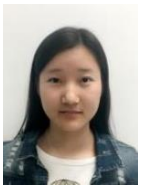
[43] Dai WY, Yang Q, Xue GR, Yu Y. Boosting for transfer learning, In: Proc. of the International Conference on Machine Learning. 2007. 193-200.

附中文参考文献:

[14]邱天宇, 申富饶, 赵金熙. 自组织增量学习神经网络综述. 软件学报, 2016,27(9):2230-2247. http://www.jos.org.cn/1000-9825/5068.htm [doi: 10.13328/j.cnki.jos.005068]

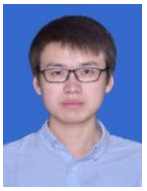
[15]王立杰,李萌,蔡斯博,李戈,谢冰,杨芙清. 基于网络信息搜索的 Web Service 文本描述信息扩充方法. 软件学报, 2012, 23(6):1335-1349. [10.3724/SP.J.1001.2012.04088]

张天伦(1991-),男,河北保定人,硕士毕业于河北大学。目前在大连海事大学攻读博士,主要研究领域为机器学习,软件工程和计算视觉.
E-mail: tlzhang@dlmu.edu.cn



祝宏玉(1994-),女,硕士研究生,毕业于深圳大学,研究领域为:神经网络,大数据算法.
E-mail:threekingdomst@163.com

陈荣(1969-),男,博士,教授,博士生导师,主要研究领域为机器学习,软件故障诊断,行为识别和运筹学.
E-mail: rchen@dlmu.edu.cn



杨溪(1993-),男,山东淄博人,硕士研究生,主要研究领域为机器学习,计算视觉和模糊集.
E-mail: yokiqust@dlmu.edu.cn

附录:

用 *softmax* 的输出来表示后验概率的合理性的说明:

定义 $x(1)\cdots x(m)$ 是 m 个输入数据,其中每个 $x(i)$ 是一个 n 维向量, $i=1\cdots m$. $x(i)_j$ 是向量第 j 维属性值. $y(1)\cdots y(m)$ 是 m 个标签值, $y(i)\in\{1\cdots k\}$, k 为类别个数.定义 $A(u,v)$ 是一个指示函数, $\forall u,v\in\{1\cdots k\}$,

$$A(u,v)=\begin{cases}1,u=v\\0,u\neq v\end{cases}$$

我们定义 $\pi()$ 是一个概率函数,即, $\pi():\mathbb{R}^n\to\mathbb{R}^k$, $\pi(x(i))_v$ 是对 $y(i)=v$ 的概率估计.

$\pi()$ 应当满足以下性质:

$$\pi(x)_v\geq 0 \tag{A.1}$$

$$\sum_{v=1}^{k}\pi(x)_v=1 \tag{A.2}$$

$$\arg\max\pi(x(i))=y(i) \tag{A.3}$$

$$\sum_{i=1}^{m}\pi(x(i))_u x(i)_j=\sum_{i=1}^{m}A(u,y(i))x(i)_j \tag{A.4}$$

根据信息论, $\pi()$ 的熵可以定义为:

$$-\sum_{v=1}^{k}\sum_{i=1}^{m}\pi(x(i))_v\log(\pi(x(i))_v) \tag{A.5}$$

在 A.1,A.2,A.4 的约束下,根据拉格朗日乘子法有:

$$\begin{aligned}L=&\sum_{j=1}^{n}\sum_{v=1}^{k}\lambda_{v,j}\left(\sum_{i=1}^{m}\pi(x(i))_v x(i)_j-A(v,y(i))x(i)_j\right)\\&+\sum_{v=1}^{k}\sum_{i=1}^{m}\beta_i(\pi(x(i))_v-1)-\sum_{v=1}^{k}\sum_{i=1}^{m}\pi(x(i))_v\log(\pi(x(i))_v)\end{aligned} \tag{A.6}$$

通过求偏导有:

$$\begin{aligned}\frac{\partial}{\partial\pi(x(i))_u}L&=\lambda_u\cdot x(i)+\beta_i-\log(\pi(x(i))_u)-1\\&=0\end{aligned} \tag{A.7}$$

根据 A.7 有:

$$\pi(x(i))_u=e^{\lambda_u\cdot x(i)+\beta_i-1} \tag{A.8}$$

根据 A.2 和 A.8 有:

$$\sum_{v=1}^{k}e^{\lambda_v\cdot x(i)+\beta_i-1}=1 \tag{A.9}$$

所以

$$e^{\beta_i}=\frac{1}{\sum_{v=1}^{k}e^{\lambda_v\cdot x(i)-1}} \tag{A.10}$$

将 A.10 带入 A.8 得

$$\pi(x(i))_u=\frac{e^{\lambda_u\cdot x(i)}}{\sum_{v=1}^{k}e^{\lambda_v\cdot x(i)}}$$

由上面的分析可以得出， $\pi()$ 的概率估计具有 softmax 的函数形式.

实验一的参数设置：

数据集	代价矩阵	隐层节点数
Eclipse_CDT_cdt-core	[0,5;2,0]	500
Eclipse_JDT_Core	[0,3;1,0]	1000
Eclipse_JDT_Debug	[0,3,1,0]	500
Eclipse_PDE_UI	[0,3;1,0]	500
Eclipse_Platform_Debug	[0,5;2,0]	200
Eclipse_Platform_SWT	[0,5;2,0]	1500
Eclipse_Platform_UI	[0,3;1,0]	2000
GNOME_ekiga_general	[0,5,2,0]	500
GNOME_Evolution_Calendar	[0,5;2,0]	1000
GNOME_Evolution_Contacts	[0,5;2,0]	1000
GNOME_Evolution_Mailer	[0,2;1,0]	3000
GNOME_Evolution_Shell	[0,3;1,0]	500
GNOME_gnome-panel_Panel	[0,5;2,0]	500
GNOME_gnome-terminal_general	[0,5;2,0]	1500
Moizlla_Core_Printing	[0,5;1,0]	500
Moizlla_Core_XPCOM	[0,5;2,0]	500
Moizlla_Core_XPConnect	[0,2;1,0]	100
Moizlla_Core_XUL	[0,5;2,0]	1000
Moizlla_Thunderbird_General	[0,5;2,0]	1000
Mozilla_Core_Layout	[0,5;2,0]	1500
Mozilla_Core_Security_UI	[0,5;2,0]	400

实验二的参数设置：

数据集	代价矩阵	隐层节点数	k 值
Eclipse_CDT_cdt-core	[0,5;2,0]	200	50
Eclipse_JDT_Core	[0,6;1,0]	200	50
Eclipse_JDT_Debug	[0,3;1,0]	200	10
Eclipse_PDE_UI	[0,3;1,0]	100	10
Eclipse_Platform_Debug	[0,5;2,0]	200	10
Eclipse_Platform_SWT	[0,3;2,0]	1000	50
Eclipse_Platform_UI	[0,5;2,0]	500	5
GNOME_ekiga_general	[0,5;2,0]	500	50
GNOME_Evolution_Calendar	[0,3;1,0]	1000	50
GNOME_Evolution_Contacts	[0,3;1,0]	500	20
GNOME_Evolution_Mailer	[0,5;2,0]	2500	200
GNOME_Evolution_Shell	[0,3;1,0]	200	50
GNOME_gnome-panel_Panel	[0,3;1,0]	500	50
GNOME_gnome-terminal_general	[0,3;1,0]	500	50
Moizlla_Core_Printing	[0,3;1,0]	500	50
Moizlla_Core_XPCOM	[0,3;1,0]	200	10
Moizlla_Core_XPConnect	[0,2;1,0]	100	5
Moizlla_Core_XUL	[0,5;2,0]	200	50
Moizlla_Thunderbird_General	[0,5;2,0]	500	50
Mozilla_Core_Layout	[0,5;2,0]	1500	50
Mozilla_Core_Security_UI	[0,5;2,0]	200	50

实验三的参数设置：

数据集	被迁移数据集	代价矩阵	隐层节点数	k值
Eclipse_CDT_cdt-core	Eclipse_JDT_Core	[0,5;2,0]	20	10
Eclipse_JDT_Core	Eclipse_JDT_Debug	[0,5;2,0]	20	10
Eclipse_JDT_Debug	Eclipse_Platform_Debug	[0,5;2,0]	200	10
Eclipse_PDE_UI	Eclipse_Platform_Debug	[0,5;2,0]	50	10
Eclipse_Platform_Debug	Eclipse_Platform_SWT	[0,5;2,0]	200	10
Eclipse_Platform_SWT	Eclipse_CDT_cdt-core	[0,5;2,0]	100	10
Eclipse_Platform_UI	Eclipse_PDE_UI	[0,5;2,0]	20	10
GNOME_ekiga_general	GNOME_Evolution_Calendar	[0,5;2,0]	100	50
GNOME_Evolution_Calendar	GNOME_Evolution_Shell	[0,5;2,0]	200	20

GNOME_Evolution_Contacts	GNOME_Evolution_Shell	[0,5;2,0]	200	10
GNOME_Evolution_Mailer	GNOME_gnome-terminal_general	[0,3;1,0]	20	10
GNOME_Evolution_Shell	Moizlla_Core_Printing	[0,5;2,0]	200	10
GNOME_gnome-panel_Panel	GNOME_ekiga_general	[0,3;1,0]	50	10
GNOME_gnome-terminal_general	GNOME_gnome-panel_Panel	[0,5;2,0]	100	50
Moizlla_Core_Printing	Moizlla_Core_XPCOM	[0,5;2,0]	20	10
Moizlla_Core_XPCOM	Moizlla_Core_Printing	[0,5;2,0]	20	10
Moizlla_Core_XPConnect	Moizlla_Core_XUL	[0,5;2,0]	20	10
Moizlla_Core_XUL	Moizlla_Core_XPConnect	[0,5;2,0]	20	10
Moizlla_Thunderbird_General	Moizlla_Core_XUL	[0,5;2,0]	30	10
Mozilla_Core_Layout	Moizlla_Core_XPCOM	[0,5;2,0]	20	10
Mozilla_Core_Security_UI	Moizlla_Core_XPCOM	[0,5;2,0]	20	10

序号	RBM 节点数(原数据集)	RBM 节点数(被迁移数据集)
1	100-500-100	100-200-100
2	100-200-100	100-200-100
3	100-500-100	100-500-100
4	100-500-200	100-1000-200
5	100-500-100	100-500-100
6	500-500-500	500-500-500
7	100-500-200	100-200-200
8	100-500-200	100-200-200
9	500-500-100	500-500-100
10	500-500-100	500-500-100
11	300-300-200	300-300-200
12	100-200-100	100-200-100
13	100-200-100	500-500-100
14	100-200-100	100-200-100
15	100-500-200	100-200-200
16	100-500-200	100-200-200
17	100-500-200	100-200-200
18	100-500-200	100-200-200
19	100-500-200	100-200-200
20	100-500-200	100-200-200
21	100-500-200	100-200-200

试验四的参数设置：

数据集	代价矩阵	隐层节点数	k 值
Eclipse_CDT_cdt-core	[0,5;2,0]	200	10
Eclipse_JDT_Core	[0,6;1,0]	200	10
Eclipse_JDT_Debug	[0,3;1,0]	200	20
Eclipse_PDE_UI	[0,3;1,0]	100	10
Eclipse_Platform_Debug	[0,5;2,0]	200	10
Eclipse_Platform_SWT	[0,3;2,0]	1000	50
Eclipse_Platform_UI	[0,5;2,0]	300	20
GNOME_ekiga_general	[0,5;2,0]	500	20
GNOME_Evolution_Calendar	[0,3;1,0]	1000	20
GNOME_Evolution_Contacts	[0,3;1,0]	500	20
GNOME_Evolution_Mailer	[0,5;2,0]	2500	100
GNOME_Evolution_Shell	[0,5;2,0]	200	10
GNOME_gnome-panel_Panel	[0,3;1,0]	500	50
GNOME_gnome-terminal_general	[0,5;2,0]	500	10
Moizlla_Core_Printing	[0,3;1,0]	500	10
Moizlla_Core_XPCOM	[0,2;1,0]	200	5
Moizlla_Core_XPConnect	[0,2;1,0]	200	5
Moizlla_Core_XUL	[0,5;2,0]	200	20
Moizlla_Thunderbird_General	[0,5;2,0]	500	20
Mozilla_Core_Layout	[0,5;2,0]	1500	50
Mozilla_Core_Security_UI	[0,5;2,0]	200	20