软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2014,25(9):1982−1991 [doi: 10.13328/j.cnki.jos.004639]

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

基于代价敏感多标记学习的开源软件分类*

韩 乐, 黎 铭

(计算机软件新技术国家重点实验室(南京大学),江苏 南京 210023)

通讯作者: 黎铭, E-mail: lim@nju.edu.cn

摘 要: 随着开源软件数量的增多,从开源软件社区中有效检索到所需的开源软件是具有挑战性的工作.现有方法通常是:首先,人工给每个软件赋予多个描述其功能、用途的标注;然后,通过关键词匹配寻找用户所需的软件.由于其简单、方便,基于标注进行软件检索得到了广泛的应用.然而,用户通常不愿意主动为其上载的开源软件提供标注,这使得根据用户上载软件的文字描述信息,从众多备选软件标注中为其自动选择能够表征其功能、用途的标注,成为了有效检索该软件的关键.把开源软件自动标注形式化为一个代价敏感多标记学习问题,并提出了一种新型代价敏感多标记学习方法 ML-CKNN.该方法通过在多标记学习中引入代价信息,有效缓解了对每一个标注而言具有该标注的示例与不具有该标注的示例分布非均衡性给多标记学习造成的影响.在 3 个开源软件社区上的实验结果表明:所提出的 ML-CKNN 方法能够为新上载的开源软件提供高质量的标注,其标注性能显著优于现有方法.

关键词: 软件挖掘;机器学习;多标记学习;代价敏感学习;软件自动标注

中图法分类号: TP311

中文引用格式: 韩乐,黎铭.基于代价敏感多标记学习的开源软件分类.软件学报,2014,25(9):1982−1991. http://www.jos.org.cn/1000-9825/4639.htm

英文引用格式: Han L, Li M. Open source software classification using cost-sensitive multi-label learning. Ruan Jian Xue Bao/ Journal of Software, 2014,25(9):1982−1991 (in Chinese). http://www.jos.org.cn/1000-9825/4639.htm

Open Source Software Classification Using Cost-Sensitive Multi-Label Learning

HAN Le, LI Ming

(National Key Laboratory for Novel Software Technology (Nanjing University), Nanjing 210023, China)

Corresponding author: LI Ming, E-mail: lim@nju.edu.cn

Abstract: With the explosive growth of open source software, retrieving desired software in open source software communities becomes a great challenge. Tagging open source software is usually a manual process which assigns software with several tags describing its functions and characteristics. Users can search their desired software by matching the keywords. Because of the simplicity and convenience, software retrieval based on tags has been widely used. However, since human effort is expensive and time-consuming, developers are not willing to tag software sufficiently when uploading software projects. Thus automatic software tagging, with tags describing functions and characteristics according to software projects' text descriptions provided by users, becomes key to effective software retrieval. This article formalizes this problem as a multi-label learning problem and proposes a new multi-label learning method ML-CKNN which can effectively solve this problem when the number of different tags is extremely large. By imposing cost value of wrong classification into multi-label learning, ML-CKNN can effectively solve this imbalanced problem, as each tag instances associated with this tag are much less than those not associated with this tag. Experiments on three open source software community datasets show that ML-CKNN can provide high-quality tags for new uploading open source software while significantly outperforming existing methods.

Key words: software mining; machine learning; multi-label learning; cost-sensitive learning; software automatic tagging

* 基金项目: 国家自然科学基金(61272217, 61321491); 教育部新世纪优秀人才计划(NCET-13-0275); 江苏省自然科学基金(BK20131278)

收稿时间: 2014-03-24; 定稿时间: 2014-05-14

近年来,开源软件得到了越来越广泛的使用.通常一个开源软件社区有几十万的开源软件,比如:开源软件社区 Ohloh 中有超过 50 万个开源软件项目,代码总量超过 200 亿行;开源软件社区 Sourceforge 中有超过 43 万个开源软件项目;开源软件社区 Freecode 中有超过 4 万个开源软件项目.全世界开源软件的总量已经超过 200 万个.如此巨大的数据量,使得用户想要正确地检索到所需的开源软件成了一个巨大的挑战.因此,许多开源社区开始使用标注(tag)进行软件检索.标注是一个语义很明确的词,它通常指软件具有某种功能或特性.用一个或一组标注就可以精确描述某种类型的开源软件,这样,用户就能够精确地检索到所需的软件.现在,给软件加标注是人工完成的.人工的过程代价很大并且很耗时,所以,软件开发者在上载软件项目时不愿意为软件提供充分多的标注.因此,为了便于用户检索开源软件,为刚上传的软件自动从备选标注中挑选合适的标注成为了关键.

已经有一些工作是关于开源软件自动标注[1,2],这些工作都是将其建模为一个标注推荐问题,首先计算出各个软件与各个标注之间的相关性,然后为每个软件推荐 k 个最相关的标注.标注数量 k 是人为设定的,比如 5 或者 10.然而,开源社区中的软件标注数量通常是不固定的,有的软件只有两三个标注,有的软件却有几十个标注.给开源软件加固定数量的标注显然是不合适的,有的软件会获得过多的标注,有的软件则会漏掉一些标注.如果能够为开源软件正确提供恰当数量的标注,则会降低数据的噪声,有利于软件检索性能的提高.

为了实现上述目标,本文把开源软件自动标注形式化为一个代价敏感多标记学习问题.图 1 是开源软件社区 Freecode 中对软件 Linux 的介绍,注意到:图上方的一段文字是对软件的总体描述,描述下面是对软件的标注,软件描述与软件标注是相互补充的[1].把从软件的描述信息中提取出属性看作示例,把软件的多个标注(tag)当作多个标记(label).ML-KNN[3]是一种有效的多标记学习方法,在很多文本的多标记分类中取得了良好的性能[4−6].然而,在开源软件自动标注问题上直接使用 ML-KNN 却难以获得较好的性能.其原因在于:开源软件社区中软件的功能和用途往往多种多样,这造成了学习器面临的备选标注集合相对较大且具有某个标注的示例往往远少于不具有该标注的示例.图 2 是 3 个开源软件社区 Freecode,Ohloh,Sourceforge 中不同标注的分布,横轴表示对于某个标签而言具有该标注的示例数量占总示例数量的比例,纵轴表示不同标注的数量.从图 2 中可以看到:对于绝大多数标注而言,具有该标注的示例数量占总示例数量比例少于 5%.这会直接造成对于每个标注而言,示例具有该标注的先验概率很小,而因为 ML-KNN 是根据后验概率来决定某个示例是否具有某个标注,所以结果导致 ML-KNN 趋向于把所有示例都预测为不具有该标注.

针对这一问题,本文提出了一种代价敏感多标记学习方法 ML-CKNN(multi-label cost-sensitive k nearest neighbors).该方法受 ML-KNN 启发,通过引入错误分类代价,让学习器更加关注对于每个标注而言具有该标注的示例,在一定程度上缓解因标注数量众多造成的示例分布不均衡给多标记学习造成的影响.在 3 个开源软件社区上的实验结果表明:本文提出的 ML-CKNN 方法能够为新上载的开源软件提供高质量的标注,其标注性能显著优于现有方法.

本文第 1 节介绍相关工作.第 2 节正式提出算法 ML-CKNN.第 3 节汇报实验结果.第 4 节总结全文.

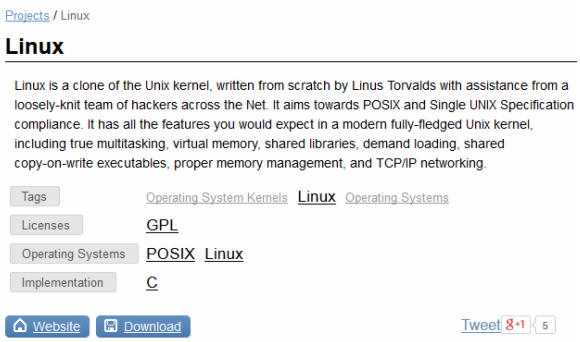


Fig.1 The profile of Linux in Freecode

图 1 Freecode 中 Linux 的介绍

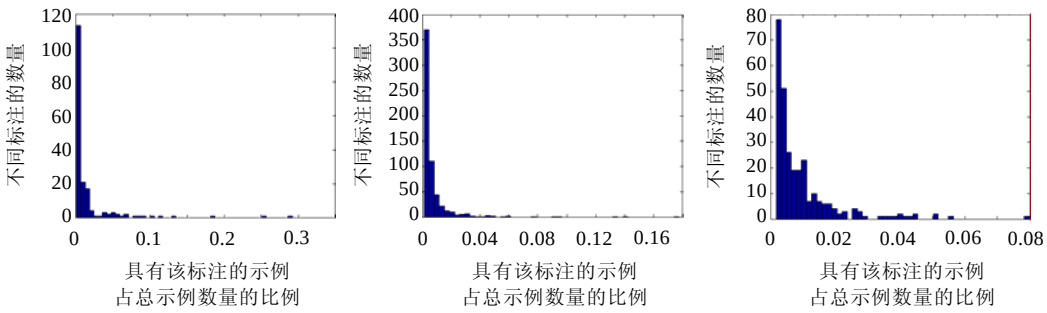


Fig.2 The tag distributions in 3 open source repositories

图 2 3 个开源软件社区中标注的分布

1 相关工作

1.1 软件标注推荐

已有一些研究工作是关于开源软件的标注推荐问题.Wang 等人[1]使用话题模型 Labeled LDA 来进行开源软件的标注推荐,在其工作中,首先利用 Labeled LDA 发现软件标注和软件描述的语义相关性,然后构建基于标注的话题与词之间的矩阵,根据产生的矩阵,使用一个简单的算法来检测软件潜在的话题,这些话题也就是软件的标注.Xia 等人[2]提出了一种软件标注推荐的方法 TagCombine,这种方法包含 3 个部分:一是多标记排序组件,这个组件把软件标注推荐看作一个多标记排序问题;二是基于相似度的标注排序组件,它使用相似的对象来推荐标注;三是基于标注-词的标注排序组件,它考虑了不同词与标注之间的关系,比如标注与词的共生关系,然后根据对象中的词推荐标注.与本文工作不同,上述工作的目标是软件标注推荐,所推荐的软件标注数量是人为提前设定的,这样通过人为设定的标注数量可能与软件真实的标注数量并不匹配,比如:人为设定推荐的标注数量通常为 5 或者 10,而软件真实的标注数量可能为 1~30 之间的任意一个数.而本文使用代价敏感的多标记分类来解决这个问题,并不需要预先设定推荐的标注数量.

1.2 多标记学习

已经有很多多标记学习算法被提出来.BR(binary relevance)算法是一种常用的问题转换算法,它为标注集合中每个不同的标注学习一个二元分类器.原始的数据集被转换为 q 个数据集,q 为标注的数量,每个标注对应一个数据集.在转换得到的数据集中,如果示例具有对应的标注,则被标记为正类;否则,被标记为负类.BR 把 q 个二元分类器的分类结果合并起来,得到最终的分类结果[7].这样,就能把一个多标记学习问题转化为一个单标记学习问题.LP(label powerset)算法使用了另外一种问题转换方式,它把训练集中每个独特的标注集合看成一类,这样多标记学习问题就转化为一个单标记多类学习问题[7].LP 算法的计算复杂度与独特标注集合的数量有关,这可能会导致复杂度问题.RAkEL 算法(random k-label sets)[8]是一个集成方法,它的基分类器是 LP 分类器,每个 LP 分类器使用标注集合的不同子集.这样,RAkEL 算法既能考虑到不同标注之间的相互关系,也能够有效避免 LP 算法的计算复杂度问题.这些方法与本文提出的 ML-CKNN 的区别在于:这些方法处理的是一般的多标记学习问题,在类别平衡的数据集上能够取得很好的效果;而本文提出的 ML-CKNN 是一个代价敏感的多标记学习方法,尤其适用于类不平衡的数据集.开源软件数据集是一个类很不平衡的数据集,对于每个标注而言,具有该标注的示例远少于不具有该标注的示例,所以适合使用 ML-CKNN 来解决开源软件分类问题.

Lo 等人[9]提出了一种代价敏感的多标记学习方法,但是他们处理的问题与本文是完全不同的.他们认为:一段音频被标注过多次,其标注具有很高的置信度;被标注过较少次的标注具有较低的置信度,可能为噪声数据.对于一段音频,他们把某个标注被标注的次数看作误分类的代价,这样,具有较高置信度的标注就不太容易被误

分类.而 ML-CKNN 要处理的是类别不平衡问题,对于每一个标注,把具有该标注的示例预测为不具有该标注会产生较大的代价;反之,则只会产生较小的代价.这样,分类器就不再趋向于把所有软件都预测为不具有该标注.

2 代价敏感的 *k* 近邻多标记分类方法 ML-CKNN

本文使用 X 表示样本空间,使用 $\boldsymbol{Y}$ 表示标注的集合.使用 T 表示训练集,S 表示测试集.训练集 T 与测试集 S 都可以看作是从一个未知分布 D 中独立同分布采样得到.多标记学习算法的任务是:根据训练集 T,训练得到一个多标记分类器 $h:X\to 2^{\boldsymbol{Y}}$,$h$ 在测试集 S 上最优化某种评价指标.

ML-CKNN 受著名的多标记学习算法 ML-KNN[3]启发,在 ML-KNN 的基础上引入了代价敏感性,并且改变了距离度量的方式,更加适合解决开源软件多标记分类问题.

使用 $\boldsymbol{y}_i$ 表示示例 x_i 所具有的标注向量,如果标注 l 为示例 x_i 的标注,则 $\boldsymbol{y}_i(l)$取值为 1;否则,$\boldsymbol{y}_i(l)$取值为 0.使用 $N(x_i)$表示示例 x_i 在训练集 T 中所具有的 k 近邻.基于示例 x_i 近邻的标注集合,可以得到一个计数向量 $\boldsymbol{C}_i(l)$,$\boldsymbol{C}_i(l)$ 表示示例 x_i 近邻中具有标注 l 的示例数量,定义如下:

$$\boldsymbol{C}_i(l)=\sum_{x_j\in N(x_i)}\boldsymbol{y}_j(l),l\in\boldsymbol{Y} \tag{1}$$

对于每个测试示例 x_i,ML-CKNN 首先使用余弦距离找到示例 x_i 在训练集 T 中的 k 近邻 $N(x_i)$.这里与 ML-KNN 使用欧几里得距离不同,余弦距离更适合本文所处理的开源软件分类问题,因为示例的属性是从文本中提取出来的.使用 F_1^l 来表示示例 x_i 具有标注 l,与其对应,使用 F_0^l 表示示例 x_i 不具有标注 l.使用 G_j^l ($j\in\{0,1,2,\ldots,k\}$)来表示示例 x_i 的 k 近邻中恰好有 j 个示例具有标注 l.根据计数向量 $\boldsymbol{C}_i(l)$,标注向量 $\boldsymbol{y}_i$ 可以用如下的最大后验概率准则得到:

$$y_i(l)=\arg\max_{b\in\{0,1\}}P(F_b^l\mid G_{\boldsymbol{C}_i(l)}^l),l\in\boldsymbol{Y} \tag{2}$$

利用贝叶斯定理,公式(2)可以写成这样的形式:

$$\boldsymbol{y}_i(l)=\arg\max_{b\in\{0,1\}}P(F_b^l)P(G_{\boldsymbol{C}_i(l)}^l\mid F_b^l)/P(G_{\boldsymbol{C}_i(l)}^l) \tag{3}$$

由于分母对最终的结果没有影响,所以公式(3)与下式等价:

$$\boldsymbol{y}_i(l)=\arg\max_{b\in\{0,1\}}P(F_b^l)P(G_{\boldsymbol{C}_i(l)}^l\mid F_b^l) \tag{4}$$

ML-CKNN 与 ML-KNN 一样都使用了贝叶斯概率,ML-KNN 直接使用了公式(4)做出最后的分类和排序,这在类别平衡的情况下能够取得很好结果;但在开源软件分类问题上却并不十分适合.在开源软件问题上,一个标注只被很少一部分软件所具有,比如在有 5 000 个软件的开源软件数据集中,很大一部分标注只被 10 个软件所具有.这样,对某个标注 l 而言,示例 x_i 的最近邻 $N(x_i)$可能全部不具有该标注,这样就不太可能预测示例 x_i 带有标注 l.除此之外,$P(F_1^l)$ 的值往往要比 $P(F_0^l)$ 小很多,即使 $P(G_{\boldsymbol{C}_i(l)}^l\mid F_1^l)$ 与 $P(G_{\boldsymbol{C}_i(l)}^l\mid F_0^l)$ 相近,分类器也趋向于把示例 x_i 预测为不具有标注 l.所以,这里不能直接使用公式(4)来做出最后的分类.

以往有很多研究工作通过改变原始训练数据的分布来引入代价敏感性[10],但这种方式在多标记学习问题中很难使用.因为对于不同的标注而言,正类样本与负类样本是不同的,也就很难进行重采样.本文的创新之处在于并没有通过重采样来引入代价敏感性,而是通过改变示例具有标注的置信度的方式来引入代价敏感性.具体来说,假设 c 为代价值,那么可以让最终示例 x_i 具有标注 l 的置信度为(即为了提高示例 x_i 被预测为具有标注 l 的可能性):

$$\theta_1^l(x_i)=cP(F_1^l)P(G_{\boldsymbol{C}_i(l)}^l\mid F_1^l)/(cP(F_1^l)P(G_{\boldsymbol{C}_i(l)}^l\mid F_1^l)+P(F_0^l)P(G_{\boldsymbol{C}_i(l)}^l\mid F_0^l)) \tag{5}$$

其中,$P(F_1^l)P(G_{\boldsymbol{C}_i(l)}^l\mid F_1^l)$ 为原始 ML-KNN 预测示例 x_i 具有标注 l 的后验概率(略去了分母,对最终结果没有影响),这里将这个值增大了 c 倍,同时对其进行了规范化,使得最终的置信度 $\theta_1^l(x_i)$ 大于等于 0 并且小于等于 1.分母中,$P(F_0^l)P(G_{\boldsymbol{C}_i(l)}^l\mid F_0^l)$ 为原始 ML-KNN 预测示例 x_i 不具有标注 l 的后验概率(略去了分母),这样提高了算法预测示例 x_i 具有标注 l 的可能性.而示例 x_i 不具有标注 l 的置信度为

$$\theta_0^l(x_i)=1-\theta_1^l(x_i) \tag{6}$$

这使得 $\theta_0^l(x_i)$ 的取值范围也在 0 到 1 之间,并且保证 $\theta_0^l(x_i)$ 与 $\theta_1^l(x_i)$ 的和为 1.如果 $\theta_1^l(x_i)\geqslant\theta_0^l(x_i)$,即 $\theta_1^l(x_i)\geqslant 0.5$,则判定示例 x_i 具有标注 l;否则,判定示例 x_i 不具有标注 l.这样取得的效果与改变原始数据的分布相似,使得最终的分类器不再趋向于把示例预测为不具有标注 l.

算法 1 是算法 ML-CKNN 的伪代码,ML-CKNN 与 ML-KNN 的算法框架基本相同,在关键的部分却有着本质的不同.ML-CKNN 的输入 T 表示训练集,k 表示用户设定的最近邻数量,x_t 表示测试示例,s 表示平滑因子,c 是用户设定的代价值.算法的输出 $\boldsymbol{y}_t$ 表示示例 x_t 的标注向量,$\boldsymbol{r}_t$ 表示标注与示例 x_t 的相关度值向量,越相关的标注值越大.算法的第(1)行、第(2)行首先计算出先验概率 $P(F_b^l)$,即对每个标注而言,具有该标注的示例与不具有该标注的示例在训练集中出现的概率.接着,第(3)行使用余弦距离在训练数据 T 中找出示例 x_i 的 k 近邻,这与 ML-KNN 使用欧几里得距离是不同的.然后,第(4)行~第(13)行计算了条件概率 $P(G_j^l\mid F_b^l)$,$b\in\{0,1\}$,$j\in\{0,1,2,\ldots,k\}$.算法的第(14)行~第(18)行计算了测试示例 x_t 的标注向量 $\boldsymbol{y}_t$ 和测试示例 x_t 对应的标注相关度值向量 $\boldsymbol{r}_t$,其中:第(14)行使用余弦距离在训练集中找出测试示例 x_t 的 k 近邻 $N(x_t)$,这与原始的 ML-KNN 是不同的.算法的第(17)行计算示例 x_t 是否具有标注 l,这里与原始的 ML-KNN 显著的不同,原始的 ML-KNN 使用最大后验概率原则计算出每个示例 x_t 是否具有标注 l;而 ML-CKNN 引入了代价敏感性,使用下面这种方式计算出示例 x_t 是否具有标注 l:

$$\boldsymbol{y}_t(l)=\begin{cases}1, \text{ if } cP(F_1^l)P(G^l_{\boldsymbol{C}_t(l)}\mid F_1^l)/(cP(F_1^l)P(G^l_{\boldsymbol{C}_t(l)}\mid F_1^l)+P(F_0^l)P(G^l_{\boldsymbol{C}_t(l)}\mid F_0^l))\geqslant 0.5\\ 0, \text{ else}\end{cases} \tag{7}$$

算法 1. MK-CKNN.

输入:

T:训练数据;k:最近邻的数量;x_t:测试示例;s:平滑因子;c:代价值;

输出:

$\boldsymbol{y}_t$:标注向量;$\boldsymbol{r}_t$:标注相关度值向量.

//首先从训练数据中计算出先验概率 $P(F_b^l)$

(1) for $l\in\boldsymbol{Y}$

(2) $P(F_1^l)=(s+\sum_{i=1}^{n}\boldsymbol{y}_i(l))/(s\times 2+n);P(F_0^l)=1-P(F_1^l)$;

//计算条件概率 $P(G_j^l\mid F_b^l)$

(3) 使用余弦距离在训练集中找出最近邻 $N(x_i)$,$i\in\{1,2,\ldots,n\}$;

(4) for $l\in\boldsymbol{Y}$

(5) for $j\in\{0,1,\ldots,k\}$

(6) $d[j]=0$; $d'[j]=0$;

(7) for $i\in\{1,2,\ldots,n\}$

(8) $\mu=\boldsymbol{C}_i(l)=\sum_{x_j\in N(x_i)}\boldsymbol{y}_j(l)$;

(9) if ($\boldsymbol{y}_i(l)$==1) then $d[\mu]=d[\mu]+1$;

(10) else $d'[\mu]=d'[\mu]+1$;

(11) for $j\in\{0,1,\ldots,k\}$

(12) $P(G_j^l\mid F_1^l)=(s+d[j])/(s\times(k+1)+\sum_{p=0}^{k}d[p])$;

(13) $P(G_j^l\mid F_0^l)=(s+d'[j])/(s\times(k+1)+\sum_{p=0}^{k}d'[p])$;

//计算示例 x_t 的标注向量 $\boldsymbol{y}_t$ 和标注相关度值向量 $\boldsymbol{r}_t$

(14) 使用余弦距离在训练集中找出最近邻 $N(x_t)$;

(15) for $l\in\boldsymbol{Y}$

(16) $\boldsymbol{C}_t(l)=\sum_{x_j\in N(x_t)}\boldsymbol{y}_j(l)$;

(17) $\boldsymbol{y}_t(l)=\begin{cases}1, \text{ if } cP(F_1^l)P(G^l_{\boldsymbol{C}_t(l)}\mid F_1^l)/(cP(F_1^l)P(G^l_{\boldsymbol{C}_t(l)}\mid F_1^l)+P(F_0^l)P(G^l_{\boldsymbol{C}_t(l)}\mid F_0^l))\geqslant 0.5\\ 0, \text{ else}\end{cases}$;

(18) $\boldsymbol{r}_t(l)=P(F_1^l\mid G^l_{\boldsymbol{C}_t(l)})=P(F_1^l)P(G^l_{\boldsymbol{C}_t(l)}\mid F_1^l)/P(G^l_{\boldsymbol{C}_t(l)})=P(F_1^l)P(G^l_{\boldsymbol{C}_t(l)}\mid F_1^l)/\sum_{b\in\{0,1\}}P(F_b^l)P(G^l_{\boldsymbol{C}_t(l)}\mid F_b^l)$;

3 实 验

3.1 数据集

如今,全世界已经有很多开源软件社区,比较著名的有 Sourceforge,Ohloh,Freecode 和 Googlecode.这些开源软件社区都包含上万甚至几十万的开源软件.每个开源软件都包括描述信息和标注信息.图 1 展示了开源软件社区 Freecode 中 Linux 的介绍,其中包括软件的文本描述信息和标注信息.注意到软件的文本描述信息和标注信息通常是相互补充的[1],本文想利用软件的文本描述信息自动地给软件加标注.正式地说,本文把每一个开源软件看作一个示例,把描述信息看作示例的属性,把标注信息看作示例的多个标记.因为开源软件的标注数量是不固定的,所以数据集中每个示例的标记数量也是不固定的.

目前还没有直接可用的开源软件数据集,我们实现了一个爬虫程序,从 Sourceforge,Ohloh,Freecode 网站上把每一个软件的文本描述信息和标注信息都爬取了下来,保存在 3 个不同的文件中.从文本中抽取出属性进行分类通常会使用 TF-IDF 属性,TF-IDF 即词频乘以逆向文件频率.我们把一个软件的描述信息看作一个文档,把不同的软件的描述信息看作不同的文档,从中抽取出 TF-IDF 属性.当然,在抽取出 TF-IDF 属性之前需要先对原始数据进行预处理,首先将描述信息中的停止词去掉,然后对描述中的词进行主干化,主干化是为了让英文中同一个词的不同形式,还原到同一个主干形式.这样不仅能降低属性的维度,也能提高分类的效果.接着,我们选择了出现频率最高的 3 000 多个词作为词集.选择出现频率较大的标注加入到标注集合,比如标注的出现频率大于 10.然后删除了描述长度过短或标注数量过少的软件,比如描述词的数量少于 20,标注的数量少于 2.最终,经过预处理后得到的 3 个开源软件数据集,见表 1.

Table 1 The preprocessed datasets

表 1 预处理后得到的数据集

数据集	词集中词的数量	标注集中标注的数量	开源软件数量
Sourceforge	3 035	283	5 734
Ohloh	3 053	603	5 100
Freecode	3 041	179	5 045

3.2 比较算法

目前已经有很多的多标记学习算法,但是几乎没有代价敏感的多标记学习算法,所以这里与一些典型的多标记学习算法进行比较,包括 BR 算法、LP 算法、RAkEL 算法[8]和原始的 ML-KNN 算法[3].BR 算法是一种流行的问题转换算法,它把多标记学习问题转换为 q 个独立的二元分类问题,这里,q 是不同标注的数量.BR 算法为每个标注训练一个分类器,如果某个示例包含标注 l,则对于标注 l 而言,这个示例是正类;否则,对于标注 l 而言,这个示例是负类.最终将 q 个分类器预测的结果合并起来作为最终的预测结果.LP 算法把训练集中每个独特的标注集合看作一个标注,然后把问题转化为单标注的多类分类问题,给定一个测试示例,LP 分类器就能输出一个最有可能的类,这个类实际上就是一组标注.RAkEL 算法是一组 LP 分类器的集成方法,每个 LP 分类器在一组小的随机标注子集上训练.这样,RAkEL 算法不仅能够考虑标注之间的相互关系,还避免了 LP 算法的性能问题[8].BR 算法、LP 算法、RAkEL 算法使用的 MULAN[12]中的实现,它们的基分类器都是线性核的支持向量机,在实验中使用的是 LibLINEAR 实现[13],参数使用的是默认参数.LibLINEAR 非常适用于高维度的文本数据分类.原始的 ML-KNN 算法与本文提出的 ML-CKNN 基本框架是相似的,不同之处在于:原始的 ML-KNN 没有引入代价敏感性,而本文提出的 ML-CKNN 方法引入了代价敏感性.ML-KNN 和 ML-CKNN 最近邻的数量 k 都设为

30,光滑因子 s 都设为 1.ML-CKNN 的代价值 c 设为 4.

3.3 评价标准

本文使用了 7 种评价标准,其中,3 种用于评价多标记分类性能,分别是基于样本的查准率(example-based precision)、基于样本的召回率(example-based recall)、基于样本的 $F1$(example-based $F1$)[7];另外 4 种是用于评价多标记学习算法的标记排序性能,分别是 1-错误率(one-error)、覆盖率(coverage)、排序损失(ranking loss)、平均精确率(average precision)[7].

这里使用 m 表示测试样本的数量,$\boldsymbol{L}$ 表示全体标注集合,$\boldsymbol{Y}_i$ 表示多标记分类器预测测试样本 x_i 具有的标注集合,$\boldsymbol{Z}_i$ 表示测试样本 x_i 实际的标注集合.

- 基于样本的查准率(example-based precision)直观上可以看作所有样本查准率的平均值,如下式所示:

$$Example\text{-}Based\ Precision=\frac{1}{m}\sum_{i=1}^{m}\frac{|\boldsymbol{Y}_i\cap\boldsymbol{Z}_i|}{|\boldsymbol{Y}_i|} \tag{8}$$

- 基于样本的召回率(example-based recall)和基于样本的 $F1$(example-based $F1$)与之类似,如下式所示:

$$Example\text{-}Based\ Recall=\frac{1}{m}\sum_{i=1}^{m}\frac{|\boldsymbol{Y}_i\cap\boldsymbol{Z}_i|}{|\boldsymbol{Z}_i|} \tag{9}$$

$$Example\text{-}Based\ F1=\frac{1}{m}\sum_{i=1}^{m}\frac{\boldsymbol{2}|\boldsymbol{Y}_i\cap\boldsymbol{Z}_i|}{|\boldsymbol{Z}_i|+|\boldsymbol{Y}_i|} \tag{10}$$

- 1-错误率(one-error)用来评价排序最高的标注有多少次不在测试示例实际的相关标注集合中,见式(11):

$$One\text{-}error=\frac{1}{m}\sum_{i=1}^{m}I(\arg\min_{\lambda\in\boldsymbol{L}}rank_i(\lambda)) \tag{11}$$

其中,$rank_i$ 表示测试样本 x_i 的标注排序,I 函数如下式所示:

$$I(\lambda)=\begin{cases}1, & \text{if } \lambda\notin\boldsymbol{Y}_i\\ 0, & \text{else}\end{cases} \tag{12}$$

- 覆盖率(coverage)是指平均需要标注排序列表的前多少个标注才能覆盖测试样本的所有标注,如下式所示:

$$Coverage=\frac{1}{m}\sum_{i=1}^{m}\max_{\lambda\in\boldsymbol{Y}_i}rank_i(\lambda)-1 \tag{13}$$

- 排序损失(ranking loss)是指平均有多少次不相关的标注排在相关的标注前面,如下式所示:

$$Ranking\ loss=\frac{1}{m}\sum_{i=1}^{m}\frac{1}{|\boldsymbol{Y}_i||\overline{\boldsymbol{Y}}_i|}|\{(\lambda_a,\lambda_b):rank_i(\lambda_a)>rank_i(\lambda_b),(\lambda_a,\lambda_b)\in\boldsymbol{Y}_i\times\overline{\boldsymbol{Y}}_i\}| \tag{14}$$

这里,$\overline{\boldsymbol{Y}}_i$ 是标注集合 $\boldsymbol{Y}_i$ 在全体标注集合 $\boldsymbol{L}$ 中的补集.

- 平均精确率(average precision)评价在某一个标注$\lambda(\lambda\in\boldsymbol{Y}_i)$之前的标注实际上也属于 $\boldsymbol{Y}_i$ 的比率.

$$Average\ precision=\frac{1}{m}\sum_{i=1}^{m}\frac{1}{|\boldsymbol{Y}_i|}\sum_{\lambda\in\boldsymbol{Y}_i}\frac{|\omega\in\boldsymbol{Y}_i:rank_i(\omega)\leqslant rank_i(\lambda)|}{rank_i(\lambda)} \tag{15}$$

开源软件数据集是类不平衡的数据集,对于每个标注而言,具有该标注的示例远少于不具有该标注的示例.一般的多标记学习算法趋向于把所有示例预测为不具有某个标注,这样分类器也能得到很高的准确率(accuracy).但是在开源软件分类问题上,分类器取得的准确率结果并不特别重要,因为即使分类器把所有的示例都预测为不具有标注,也能取得 90%以上的准确率.但是这样的预测结果是没有任何意义的,最终没有得到任何预测标注.类似的,使用海明损失作为评价标准也不合适,一个多标记分类器把所有的测试示例都预测为不具有标注,也能得到很低的海明损失.软件标注问题更关心的是分类器预测出的标注有多少实际上确实是软件的标注,软件实际上具有的标注有多少被分类器预测出来.所以在评价算法分类性能时,本文使用的是基于样本的查准率、基于样本的召回率以及基于样本的 $F1$,实验结果见表 2~表 4.此外,为了对比多标记学习的有效性,本文

还采用多标记学习中常用的 4 种评价准则——1-错误率、覆盖率、排序损失和平均准确率来对软件分类性能进行评估,结果见表 5~表 8.其中,每个单元格中的第 1 个数值表示 10 折交叉验证的均值,第 2 个数值表示 10 折交叉验证的标准差,每一行最好的结果用粗体表示.

Table 2 The example-based precision results of comparing algorithms (the larger, the better)

表 2 对比算法在基于样本查准率上的性能比较(数值越高越好)

数据集	BR	LP	RAkEL	ML-KNN	ML-CKNN
Freecode	0.419±0.015	0.359±0.014	0.427±0.014	0.296±0.031	**0.451±0.007**
Ohloh	0.393±0.017	0.159±0.015	**0.398±0.020**	0.174±0.019	0.368±0.012
Sourceforge	0.213±0.010	0.177±0.008	0.221±0.010	0.064±0.013	**0.248±0.013**

Table 3 The example-based recall results of comparing algorithms (the larger, the better)

表 3 对比算法在基于样本召回率上的性能比较(数值越高越好)

数据集	BR	LP	RAkEL	ML-KNN	ML-CKNN
Freecode	0.330±0.013	0.325±0.015	0.331±0.015	0.158±0.019	**0.518±0.017**
Ohloh	0.213±0.012	0.145±0.013	0.217±0.014	0.061±0.007	**0.295±0.011**
Sourceforge	0.158±0.009	0.166±0.009	0.161±0.010	0.031±0.006	**0.223±0.015**

Table 4 The example-based *F*1 results of comparing algorithms (the larger, the better)

表 4 对比算法在基于样本 *F*1 上的性能比较(数值越高越好)

数据集	BR	LP	RAkEL	ML-KNN	ML-CKNN
Freecode	0.344±0.012	0.329±0.014	0.348±0.013	0.193±0.021	**0.457±0.009**
Ohloh	0.254±0.013	0.141±0.012	0.258±0.015	0.083±0.009	**0.300±0.011**
Sourceforge	0.170±0.010	0.167±0.009	0.174±0.010	0.041±0.008	**0.216±0.012**

Table 5 The one-error results of comparing algorithms (the smaller, the better)

表 5 对比算法在 1-错误率上的性能比较(数值越低越好)

数据集	BR	LP	RAkEL	ML-KNN	ML-CKNN
Freecode	0.601±0.018	0.635±0.015	0.498±0.020	0.547±0.021	**0.370±0.013**
Ohloh	0.618±0.021	0.858±0.013	0.550±0.020	0.693±0.013	**0.506±0.016**
Sourceforge	0.786±0.011	0.820±0.013	0.733±0.010	0.813±0.016	**0.673±0.021**

Table 6 The coverage results of comparing algorithms (the smaller, the better)

表 6 对比算法在覆盖率上的性能比较(数值越低越好)

数据集	BR	LP	RAkEL	ML-KNN	ML-CKNN
Freecode	97.991±2.390	94.584±2.197	86.982±1.919	49.549±1.463	**30.149±1.439**
Ohloh	445.502±6.546	450.721±7.498	428.967±8.282	303.705±6.816	**208.132±7.526**
Sourceforge	176.755±4.134	175.670±2.858	168.457±4.239	102.035±2.398	**70.457±2.230**

Table 7 The ranking loss results of comparing algorithms (the smaller, the better)

表 7 对比算法在排序损失上的性能比较(数值越低越好)

数据集	BR	LP	RAkEL	ML-KNN	ML-CKNN
Freecode	0.297±0.009	0.299±0.010	0.245±0.008	0.142±0.007	**0.076±0.005**
Ohloh	0.404±0.010	0.441±0.012	0.365±0.011	0.203±0.006	**0.118±0.005**
Sourceforge	0.414±0.011	0.412±0.009	0.379±0.011	0.214±0.007	**0.133±0.005**

Table 8 The average precision results of comparing algorithms (the larger, the better)

表 8 对比算法在平均精确率上的性能比较(数值越高越好)

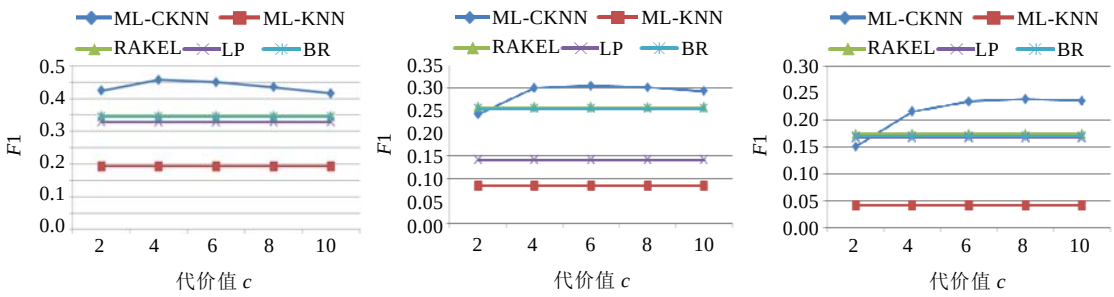
数据集	BR	LP	RAkEL	ML-KNN	ML-CKNN
Freecode	0.301±0.011	0.310±0.013	0.391±0.013	0.394±0.013	**0.555±0.009**
Ohloh	0.186±0.010	0.107±0.010	0.239±0.013	0.203±0.008	**0.344±0.010**
Sourceforge	0.159±0.008	0.159±0.009	0.207±0.009	0.191±0.010	**0.309±0.013**

3.4 实验结果

表 2~表 4 中的实验结果表明:ML-CKNN 在 3 个数据集上都取得了最好的基于样本的召回率和 *F*1 结果,

在 2 个数据集上取得了最好的基于样本的查准率结果.值得一提的是:从表 5~表 8 所给出的 4 种多标记学习常用的评价准则上,ML-CKNN 方法都明显优于所有对比方法.特别地,本文提出的代价敏感多标记学习方法 ML-CKNN 在开源软件分类问题上要显著好于非代价敏感的 ML-KNN,这表明:在开源软件分类问题上合理引入代价增加学习器对具有标注的示例的敏感性,能够有效提升学习性能.

下面讨论代价值 c 对 ML-CKNN 性能的影响.图 3 是 ML-CKNN 取得的基于样本 $F1$ 结果随代价值 c 的变化情况.图 3(a)是在 Freecode 数据集上的结果,图 3(b)是在 Ohloh 数据集上的结果,图 3(c)是在 Sourceforge 数据集上的结果.从图 3 中可以看到:当代价值 c 在 4~6 之间时,ML-CKNN 能取得最高的基于样本 $F1$ 结果.所以,在实验中设置代价值 c 为 4.另外,代价值 c 在 2~10 之间变化的过程中,ML-CKNN 在绝大部分情况下都取得了最高的基于样本 $F1$ 结果.



(a) Freecode (b) Ohloh (c) Sourceforge

Fig.3 The example-based F1 results of ML-CKNN with different cost value c

图 3 ML-CKNN 取得的基于样本 $F1$ 结果随代价值 c 的变化

4 结束语

准确有效地为上载至开源软件社区的软件提供描述其功能、用途、特性等的标注,是帮助用户快速寻找所需软件的关键.本文把开源软件自动标注形式化为一个代价敏感多标记学习问题,并提出了一种代价敏感多标记学习方法——ML-CKNN.该方法通过引入错误分类代价,有效缓解了对于每个标注而言,示例分布非均衡性给多标记学习带来的影响.实验结果表明,ML-CKNN 能够获得较对比方法更优的软件自动标注性能.

在将来的工作中,可以使用度量学习的方式来学习一种度量,使得具有相同标注的样本在这种度量中尽可能靠近,具有不同标注的样本在这种度量中尽可能远.此外,利用不同标注之间的相互关系来帮助进一步提升学习性能,也值得在将来的工作中进一步探索.

References:

[1] Wang T, Yin G, Li X, Wang H. Labeled topic detection of open source software from mining mass textual project profiles. In: Proc. of the 1st Int'l Workshop on Software Mining. New York: ACM Press, 2012. 17–24. [doi: 10.1145/2384416.2384419]

[2] Xia X, Lo D, Wang X, Zhou B. Tag recommendation in software information sites. In: Proc. of the 10th Int'l Workshop on Mining Software Repositories. Piscataway: IEEE Press, 2013. 287–296.

[3] Zhang ML, Zhou ZH. ML-KNN: A lazy learning approach to multi-label learning. Pattern Recognition, 2007,40(7):2038–2048. [doi: 10.1016/j.patcog.2006.12.019]

[4] De Souza AF, Pedroni F, Oliveira E, Ciarelli PM, Henrique WF, Veronese L, Badue C. Automated multi-label text categorization with VG-RAM weightless neural networks. Neurocomputing, 2009,72(10):2209–2217. [doi: 10.1016/j.neucom.2008.06.028]

[5] Ciarelli PM, Oliveira E, Badue C, De Souza AF. Multi-Label text categorization using a probabilistic neural network. Int'l Journal of Computer Information Systems and Industrial Management Applications, 2009,1:133–144.

[6] Jiang JY, Tsai SC, Lee SJ. FSKNN: Multi-Label text categorization based on fuzzy similarity and *k* nearest neighbors. Expert Systems with Applications, 2012,39(3):2813−2821. [doi: 10.1016/j.eswa.2011.08.141]

[7] Tsoumakas G, Katakis I, Vlahavas I. Mining multi-label data. In: Data Mining and Knowledge Discovery Handbook. 2010. 667−685. [doi: 10.1007/978-0-387-09823-4_34]

[8] Tsoumakas G, Vlahavas I. Random *k*-label sets: An ensemble method for multilabel classification. In: Proc. of the 18th European Conf. on Machine Learning. Berlin, Heidelberg: Springer-Verlag, 2007. 406−417. [doi: 10.1007/978-3-540-74958-5_38]

[9] Lo HY, Wang JC, Wang HM, Lin SD. Cost-Sensitive multi-label learning for audio tag annotation and retrieval. IEEE Trans. on Multimedia, 2011,13(3):518−529. [doi: 10.1109/TMM.2011.2129498]

[10] Liu AY. The effect of oversampling and undersampling on classifying imbalanced text datasets [Ph.D. Thesis]. Austin: The University of Texas at Austin, 2004.

[11] Yang Y. An evaluation of statistical approaches to text categorization. Information Retrieval, 1999,1(1-2):69−90. [doi: 10.1023/A:1009982220290]

[12] Tsoumakas G, Spyromitros-Xioufis E, Vilcek J, Vlahavas I. MULAN: A Java library for multi-label learning. The Journal of Machine Learning Research, 2011,12:2411−2414.

[13] Fan RE, Chang KW, Hsieh CJ, Wang XR, Lin CJ. LIBLINEAR: A library for large linear classification. The Journal of Machine Learning Research, 2008,9:1871−1874.

[14] Tsoumakas G, Katakis I. Multi-Label classification: An overview. Int'l Journal of Data Warehousing and Mining, 2007,3(3):1−13. [doi: 10.4018/jdwm.2007070101]

[15] Trohidis K, Tsoumakas G, Kalliris G, Vlahavas IP. Multi-Label classification of music into emotions. In: Proc. of the 9th Int'l Conf. on Music Information Retrieval. Philadelphia, 2008. 325−330.

韩乐(1990－),男,江苏兴化人,主要研究领域为软件挖掘,机器学习.
E-mail: hanl@lamda.nju.edu.cn

黎铭(1980－),男,博士,副教授,CCF 会员,主要研究领域为软件挖掘,机器学习.
E-mail: lim@nju.edu.cn