BUTTER: 一种基于主题模型和异构网络的缺陷分发方法①

韩广乐, 张 文, 王 青

(中国科学院软件研究所 互联网软件技术实验室, 北京 100190)

摘 要: 当软件缺陷被提交到缺陷跟踪系统并经过确认之后, 它会被分发给开发人员进行缺陷修复. 这个过程就叫做缺陷分发. 随着被提交到系统的缺陷报告日益增多, 手工分发缺陷报告会变得越来越困难. 提出了一种自动分发缺陷的方法 BUTTER. 与其他方法不同的是, BUTTER 不仅利用主题模型分析缺陷报告中的文本信息, 而且创新性地建立了一个包含提交者、缺陷和开发者三种节点及其相互关系的异构网络, 从该异构网络中抽取了更多的结构信息. 实验证明, BUTTER 进行自动缺陷分发较其他缺陷自动分发方法要好.

关键词: 缺陷分发; 主题模型; 异构网络

BUTTER: An Approach to Bug Triage with Topic Modeling and Heterogeneous Network Analysis

HAN Guang-Le, ZHANG Wen, WANG Qing

(Laboratory for Internet Software Technologies, Institute of Software, Chinese Academy of Sciences, Beijing 100190, China)

Abstract: When a bug was reported to the bug tracking system, it should be assigned to a developer who is responsible for its resolution after it is confirmed. This processing is called bug triage. With increasing number of bug reports submitted to the bug tracking system, it is more and more difficult to assign appropriate developers to the reported bugs manually. In this paper, we propose an approach called BUTTER (BUg Triage by topic modeling and heTERogeneous network analysis) to automatically assign bugs to developers. Different from existing work, BUTTER not only uses topic model to analyze the text information from bug reports, but also innovatively takes structural information into consideration by constructing a heterogeneous network which includes relationships among submitters, bugs and developers. Experiment shows that BUTTER outperforms other methods on automated bug triage.

Key words: bug triage; topic model; heterogeneous network

1 引言

Bugzilla、JIRA 等开源缺陷跟踪系统已经在软件开发和维护过程中被广泛采用. 它为分布在世界各地的开发者提供了一个提交缺陷、讨论问题的平台, 极大地增加了发现和修改缺陷的效率, 提高了软件开发的质量[1].

然而, 随之而来的问题是, 每天都有成百份的缺陷报告被提交到系统中, 并且每一份缺陷报告都需要被分发给一个合适的开发者进行修改[2], 这个过程叫做缺陷分发. 负责缺陷分发的人员需要分析每份缺陷报告并在所有开发者中选择最适合的开发者去修改缺陷, 这个工作量无疑是巨大的. Jeong et al.[3]研究发现在 Eclipse 项目中每个缺陷平均需要 40 天来完成分发.

为了减少缺陷分发的工作量, 研究人员提出了一些自动化或半自动化的方法[4,5]. 这些方法的主要思路如下: 首先, 收集一个时期内的缺陷报告数据, 提取缺陷报告中总结和缺陷描述等文字部分作为文档. 然后, 利用自然语言处理和机器学习方法在文档和开

① 基金项目:国家自然科学基金(71101138,91218302,61379046,91318301);北京市自然科学基金(4122087)
收稿时间:2014-02-19;收到修改稿时间:2014-03-24

发者之间建立起联系, 形成稳定的模型. 最后, 当新的缺陷报告被提交后, 利用建立好的模型就可以自动推荐合适的开发者去修复缺陷.

在上述方法中, 缺陷报告中的文本描述的质量对于自动化缺陷分发的结果是决定性的因素. 文本质量高的缺陷会很容易的分发, 而文本质量低的缺陷则难以及时分发. 由于不同的缺陷提交者对缺陷有不同的理解, 且书写风格也不尽相同, 有些缺陷描述会含糊不清. 再加上掺杂在自然语言中的代码片段和链接, 导致文本质量会比较低, 这是导致缺陷自动化分发方法失败的主要原因.

我们发现, 除了文本信息之外, 在缺陷报告中还包含有结构信息. 如果我们将缺陷提交者、缺陷和开发者视为节点, 提交关系、评论关系视为它们之间的边, 这就形成了一个异构网络. 异构网络中蕴含了丰富的结构信息, 是对文本信息的一个重要补充. 本文提出了一种结合了文本信息和结构信息的自动化缺陷分发算法 BUTTER. 虽然每个缺陷报告上只记录了一位修复者, 但是我们认为缺陷修复过程是许多开发者在评论中不断讨论共同合作的结果[6], 因此 BUTTER 的目标是综合利用缺陷报告中的文本信息和结构信息以找到有可能对特定缺陷有兴趣的开发者.

接下来, 第 2 节将对研究问题涉及的相关概念进行阐述, 第 3 节具体描述 BUTTER 算法的过程, 第 4 节给出具体实验结果及讨论, 最后是工作总结和未来研究的展望.

2 背景

2.1 主题模型

主题模型是一种靠分析文档集中词语的共现情况来发现文档集中抽象主题的统计模型. LDA[7]就是一种典型的主题模型. LDA 是一个产生式贝叶斯概率模型, 它通过分析离散词语来挖掘文档中的潜在结构. LDA 认为每一个文档都是由有限个主题混合而成的, 而每一个主题在文档词汇中都有自己的概率分布. 每一个文档都有一个在所有主题上的概率分布的向量, 向量中所有元素加和为 1.我们称这个向量为主题分布向量.

2.2 异构网络

一般来说, 信息网络指的是一个图 $G = (V, E)$, 图中有 T 种不同的节点 $X = \{X_t\}_{t=1}^{T}$. X_t 表示的是一组属于 t_{th} 类型的节点. V 表示所有的节点, 即 $V = \bigcup_{t=1}^{T} X_t$. E 表示 V 中任意两个节点之间的边. 当 T=1 时, 这个网络就是一个同构网络; 当 $T \geq 2$ 时, 这个网络就是一个异构网络.

在 BUTTER 算法中, 我们采用了 Ji et al.[8]提出的 RankClass 算法来抽取异构网络中的结构信息. 利用异构网络中的结构信息, RankClass 能同时对节点进行分类和排序. 与 LDA 相对应, RankClass 能将节点分类到不同的主题下. 不同于同构网络中的排序算法, RankClass 将所有的节点在每一个主题下分别排序. 在本文中引入 RankClass 算法是合适的, 不同的开发者有不同主题的兴趣和特长, 每一个主题下以概率的方式排序. 一个开发者在一个主题下有一个概率值, 表示了此开发者在该主题下的熟练度, 我们称它为专长得分. 在同一个主题下, 所有开发者的专长得分之和为 1.

3 BUTTER算法

图 1 展示了 BUTTER 算法的框架. 虚线部分是训练过程, 实线部分则展示了处理一个新缺陷的过程. 首先, 我们用之前修复过的缺陷中的文本信息来训练 LDA 主题模型. 然后, 基于训练出的主题模型和从缺陷报告中提取的结构信息, 训练出 RankClass 模型. 当收到一个新缺陷时, 我们首先用 LDA 和 RankClass 模型计算它的主题分布. 缺陷最终的主题分布和开发者在这些主题上的专长得分将最终决定候选开发者. 以下将分四个部分详细论述 BUTTER 算法.

3.1 LDA 建模

缺陷有许多不同的状态. 参考已有研究 Anvik et al.[9]的做法, 我们只利用缺陷状态为 RESOLVED、VERIFIED 或 CLOSE 且解决状态为 FIXED 的有效缺陷进行建模. 从缺陷报告中抽取总结和缺陷描述部分作为训练集, 并在预处理部分完成分词、去停用词、去非词典词等步骤.

之后, 我们使用 GibbsLDA++[10]开源工具进行建模. LDA 算法需要设定三个参数: 主题个数 K, 超参数 α 和 β. 这里我们使用工具默认值 α=50/K, β=0.01. 目前在 LDA 建模中没有一种方法能预知对于文档集来说最合适的主题个数, 因为这与文档长度、文档中词汇分布等其他因素有关, 不同的文档有不同的最佳主题个数. 我们依次尝试了许多不同的 K, 从中选择结果最好的那个. 在接下来对 Eclipse JDT 数据的实验中发现 K=10 的情况下能取得最优自动分发效果.

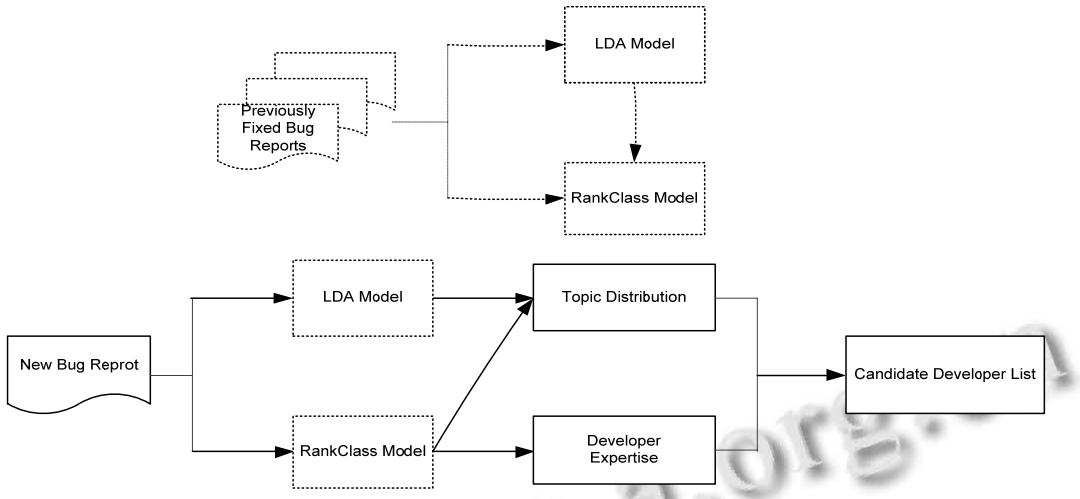

图 1 BUTTER 算法的框架

GibbsLDA++通过 Gibbs 采样的方法学习 LDA 模型. 初始时, 随机给每一个缺陷报告中的每一个词 i 分配一个主题 z_i (z_i 取值为 1 到 K), 然后统计每个词 t 在主题 k 下出现的次数 $n_k^{(t)}$ 和每个缺陷报告 m 中属于主题 k 的词的个数 $n_m^{(k)}$. 然后, 对于每一个词 i, 利用除 i 以外的词的主题分配估计 i 的各个主题概率, 如公式(1)所示[14]. 当得到 i 的所有主题概率分布后, 按照这个概率分布为其采样一个新的主题. 采用同样的方法不断更新下一个词的主题, 直到每一个缺陷报告的主题分布都收敛. 此时, 缺陷报告 m 属于第 k 个主题的概率 $\theta_{m,k}$ 可以由公式(2)计算得出.

$$p(z_i = k \mid z_{-i}) \propto \frac{n_{k,-i}^{(t)} + \beta_t}{\sum_{t=1}^{V} n_{k,-i}^{(t)} + \beta_t} * \frac{n_{m,-i}^{(k)} + \alpha_k}{\left[\sum_{k=1}^{K} n_m^{(k)} + \alpha_k\right] - 1} \quad (1)$$

$$\theta_{m,k} = \frac{n_m^{(k)} + \alpha_k}{\sum_{k=1}^{K} n_m^{(k)} + \alpha_k} \quad (2)$$

训练完成之后, 我们利用训练的模型来推导一个新缺陷的主题分布. 表 1 是一个主题分布的例子. 在表 1 中我们设定总共有四个主题, 列出了四个缺陷的主题分布. 从中我们可以看出, 缺陷 1 最有可能属于主题 2, 缺陷 3 和缺陷 4 很明显分别属于主题 1 和主题 3, 但是缺陷 2 在主题 1、主题 3 和主题 4 上分布较为平均.

3.2 异构网络分析

从图 1 中可以看出, RankClass 模型的输入有两部分, 一是从历史缺陷报告中抽取出的异构网络, 二是从 LDA 模型中选取的标有属于某个特定主题的种子缺陷.

表 1 主题分布示例

	Topic 1	Topic 2	Topic 3	Topic 4
Bug 1	0.1	0.6	0.2	0.1
Bug 2	0.3	0.1	0.2	0.4
Bug 3	0.7	0.1	0.1	0.1
Bug 4	0.1	0.2	0.6	0.1

在 BUTTER 算法中, 使用的异构网络中含有三种节点: 提交者、缺陷和开发者. 图 2 是从 Eclipse JDT 中选取的一个异构网络的示例. 在这个例子中, Adam Kiezun 是提交者, 他提交了缺陷 9673 和缺陷 10602. Adam Kiezun、Andre Weinand 和 Erich Gamma 作为开发者参与了缺陷 9673 的修改, 而 Erich Gamma 和 Darin Swanson 参与了缺陷 10602 的修改. 缺陷与开发者之间边的权重表示了该开发者在该缺陷上的评论数, 而开发者之间的边的权重表示了相互之间评论的数量.

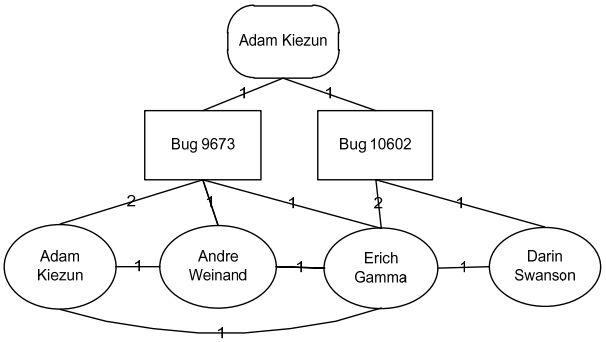

图 2 异构网络示例

由于 LDA 算法是非监督的，因此实际上并不知道每个缺陷属于的主题. Ji et al.[8]证明 RankClass 是一

个鲁棒性很强的算法, 即使在初始种子缺陷质量不是很高的情况下也能产生较为合理的结果, 因此我们使用 LDA 算法中主题归属较为明确的缺陷报告作为种子.

在一个缺陷的主题分布中, 我们选择概率最高的那个主题作为这个缺陷报告的主题. 这个主题的概率越高, 说明我们认为该缺陷报告越有可能属于这个主题. 以表 1 为例, 缺陷 1 和缺陷 2 分别属于主题 2 和主题 4. 但是我们对缺陷 1 属于主题 2 更为确定. 因此, 为了选定高质量的种子缺陷, 本文将训练集中的所有缺陷按照它属于主题的概率大小进行排序. 根据 Ji et al.[8], 本文选取了训练集中概率最高的前 0.5%的缺陷作为种子缺陷.

在获得了异构网络和种子缺陷后, RankClass 算法运行后会有两个产出结果. 一是所有提交者兴趣的主题分布, 它类似于表 1. 二是所有开发者在每个主题下的专长的得分, 表 2 是一个例子. 在表 2 中共有四个主题和五个开发者. 在每个主题下, 所有开发者的专长得分加和为 1. 开发者 2 是一个重要的开发者, 因为他在这四个主题上的的专长得分为(0.6, 0.2, 0.5, 0.2), 其中他在主题 1、主题 3 和主题 4 上都很熟悉和专业. 开发者 4 在主题 2 上是最好的, 开发者 1 在主题 4 上贡献很多. 相反的, 开发者 3 和开发者 5 在这个项目中并不是非常重要, 因为他们在这四个主题上都不擅长.

表 2 专长得分示例

	Topic 1	Topic 2	Topic 3	Topic 4
Dev 1	0.1	0.1	0.1	0.3
Dev 2	0.6	0.2	0.5	0.4
Dev 3	0.1	0.1	0.2	0.1
Dev 4	0.1	0.5	0.1	0.1
Dev 5	0.1	0.1	0.1	0.1

3.3 调整缺陷的主题分布

诸多情况表明, 当文本信息质量很差时, 仅仅使用文本信息所得到的结果也很差. 在这种情况下, 加入提出者、缺陷和开发者之间的结构信息就变得很重要了.

在建立的异构网络中, 三种节点被连接在一起. 缺陷的主题信息会通过网络传播给提出者和开发者. RankClass 算法是一个迭代算法. 每一次迭代中要计算每个节点在各个主题下的专长得分, 他们会用来调整网络中边的权重, 进而调整整个网络的结构. 反过来, 网络中边权重的调整会提高节点的专长得分精度. 经过足够的传播和迭代, RankClass 会得到一个稳定的模型. 这样, 提交者节点的主题分布就反映了缺陷节点的主题信息[8]. 因此, 我们用提交者节点的主题分布来调整 LDA 中得到的缺陷的主题分布. 最终的主题分布可以表示为:

$$\theta_{bug} = r * \theta_{\mathrm{LDA,bug}} + (1-\mathrm{r}) * \theta_{HN,sub} \quad (3)$$

在这里, θ_{bug} 是缺陷的最终主题分布, $\theta_{LDA,bug}$ 和 $\theta_{HN,sub}$ 分别是 LDA 模型中产生的缺陷的主题分布和异构网络中提交者节点的主题分布. 它们两者的权重为 r, 取值在 0 到 1 之间.

3.4 产生候选开发者列表

BUTTER 算法的最后一步就是计算每个开发者会修复某个特定缺陷的概率. 这个概率可以表示为条件概率 *P(dev|bug)*. 每个缺陷都可以被分解为几个主题, 每个开发者在每个主题下都有一个专长得分. 因此, *P(dev|bug)*可以用以下公式计算:

$$P(dev|bug) = \sum_{topic} P(\mathrm{topic}|bug) * P(dev|\mathrm{topic}) \quad (4)$$

对每一个缺陷, 我们都按照 *P(dev|bug)*概率的大小将开发者进行排序, 从中选择概率最大的 N 个开发者形成一个候选开发者列表. 在 BUTTER 算法将缺陷分配给候选开发者列表中的开发者前, 这个列表需要进行一些调整. 在开源环境下, 开发者可以在任何时间进入和退出, 因此在分配缺陷之前必须要检查列表中的开发者是否还活跃. 根据 Bhattacharya et al.[11], 我们移除了列表中 100 天内没有活动的开发者. 到这里, 我们得到了最终要推荐给缺陷的开发者列表.

4 实验结果及讨论

在这一节, 我们使用开源项目 Eclipse JDT 来对 BUTTER 算法进行实验, 并与已有 DRETOM 算法[12]进行对比. BUTTER 算法预测的是对缺陷报告感兴趣的开发者, 这里为使评价简化, 我们认为感兴趣的开发者是参与该缺陷讨论的开发人员. 由于每个缺陷报告参与评论的人数不一, 准确率将失效, 因此我们使用召回率作为实验指标. 召回率的定义如下:

$$recall = \frac{\#\text{ of appropriate recommendations}}{\#\text{ of developers participate in the bug}} \quad (5)$$

4.1 实验数据观察

我们收集了 Eclipse JDT 中从 2002 年到 2009 年的

所有有效缺陷报告, 总共包含 18674 个缺陷、3441 个开发者、2712 个缺陷提交者和 128058 条评论.

在这些数据中呈现了明显的不均衡现象. 虽然有很多开发者参与了这个项目, 但只有很少一部分人对项目有持续的贡献: 只有 83 个开发者在这 8 年内有超过 100 次的评论; 2933 个开发者参与少于 10 个缺陷, 我们认为这些开发者是不可靠的. 另外, 在总共 2712 个缺陷提出者中, 只有 170 个提出了超过 10 个缺陷, 大部分都不是这个项目的长期缺陷提出者. 大部分缺陷都是在被提交后很短时间内得到了修复. 70%的缺陷在三个月内修复, 而 90%的缺陷在一年之内修复. 大部分的缺陷的评论数目在 2 到 6 之间, 88.7%的缺陷包含少于 10 条评论.

4.2 实验准备

由于 90%的缺陷在一年之内修复, 所以我们选择 1 年作为间隔来处理数据. 从 2002 年到 2009 年, 我们共有 8 份数据. 在每一份数据中, 在 1 月到 9 月之间修复的缺陷作为训练集, 在 10 月到 12 月之间修复的缺陷作为测试集.

参与项目的开发者很多, 但大部分都不可靠. 根据前一节的数据, 将所有开发者按照评论数目从大到小排序, 保留前面贡献了全体评论的 90%的开发者, 移除剩余的不可靠的开发者及其评论. 同时, 删除那些由于删除了开发者的评论而变得没有评论的缺陷. 经过这一过滤, 我们得到了最终实验需要的数据, 具体如表 3 所示.

表 3 每份数据中训练集、测试集、开发者、提交者的数目

	Training set	Test set	Developer	Submitter
2002	2594	512	66	52
2003	1661	457	58	52
2004	1589	463	67	46
2005	1882	308	76	43
2006	1274	218	50	25
2007	987	166	39	22
2008	773	108	27	17
2009	459	75	42	11

如表 3 所示, 经过过滤后, 训练集和测试集总共有 13526 个缺陷. 不同缺陷涉及到开发者的数目不同. 在训练集中, 每个缺陷平均有 3 位开发者进行讨论和开发. 因此我们对每一个缺陷推荐平均数目的 2 倍, 即 6 位开发者.

4.3 实验结果

为了得到最合适的参数 r, 从 0 到 1, 每次间隔 0.1, 我们做了 11 组实验. 在 2004 年数据上的实验结果如表 4 所示. 从 Top 1 到 Top 6, 除了 Top 6 在 r=0.8 时较高外, 其他都是在 r=0.7 时召回率达到最高, 而且在 Top 6 与 r=0.8 的结果相差无几. 在其他年份的数据上实验结果也与之相似, 因此我们认为 r 的最合适选择是 0.7. 在以后的实验中也将 r 定为 0.7.

表 5 展示了从 2002 年到 2009 年 BUTTER 和 DRETOM 两种算法的召回率. 可以看出, BUTTER 得到的结果比 DRETOM 要好. 其中在 2002、2003、2004 和 2008 年中, BUTTER 的结果要远远好于 DRETOM, 在 2005、2006、2007 和 2009 年中, 尽管结果非常接近, 但 BUTTER 还是要稍微优于 DRETOM. 平均来说, BUTTER 将召回率提高了 5%. 特别地, 在 2002 年的 Top 6 提高了 15.72%. BUTTER 在 Top 6 的召回率除了 2005 年以外都高于 60%. 注意到 DRETOM 在 2005 年的召回率也低于其他年, 所以我们认为这个波动很可能是由实验之外的其他因素造成的.

4.4 实验结果讨论

4.4.1 引入 r 的效果

在公式(3)中, 我们引入了参数 r 来调节从 LDA 模型得到的缺陷的主题分布和从异构网络中得到的提交者的主题分布之间的权重. 当 r=0 时, 表示只使用从异构网络中得到的提交者的主题分布; 当 r=1 时, 表示只使用从 LDA 中得到的缺陷的主题分布. 当 r 介于 0 和 1 之间时, 最佳结果就是两者的按权加和. 在实验中我们发现 r=0.7 是最优选择, 这说明最终的主题分布是由两者共同按权加和确定的. r>0.5, 表示从 LDA 处得到的主题分布权重比从异构网络处得到的主题分布权重高, 由此也可推断出 Eclipse JDT 数据集上的缺陷报告中的文本质量相对较高.

在表 4 中, Top 6 的最佳召回率为 59.38%, 最差召回率为 54.69%, 它们并没有相差很多. 这似乎说明参数 r 并没有对结果产生很大影响. 我们认为有至少两个可能的原因. 第一, 我们引入参数 r 的目的就是为了调节两个主题分布, 使之更接近于缺陷本身真正的主题分布. 如果这两个分布本身就相似, 那么参数 r 确实没有起太大的作用. 在这种情况下, 尽管 r 没起太大作用, 但两个相似的主题分布使我们更确定这个分布十分接近于缺陷本身真正的主题分布, 因此最终的推荐结果也会很好. 第二, 在实验前的过滤中, 我们移除

了那些提交缺陷数很少的提交者, 而且移除的数目不少. 我们看到, 对于那些提交者已经被删除了的缺陷, 它们没有公式(3)中的第二项, 因此最终得到的主题分布是与 LDA 中得到的主题分布严格正比的. 虽然最终计算的概率不同, 但不会影响到开发者的排序, 因此在这种情况下, r 不发挥作用. 综合来看, 由于有以上两种因素的存在, 平均召回率能提升 5%已经能够说明 r 确实起到了很重要的作用.

表 4 2004 年测试集上不同参数 r 的召回率 (%)

r	Top1	Top2	Top3	Top4	Top5	Top6
0	7.66	19.37	32.34	45.34	50.35	54.69
0.1	7.84	20.31	33.24	45.34	51.82	54.76
0.2	8.03	21.22	35.84	45.03	51.79	55.30
0.3	8.26	23.74	37.54	45.46	52.24	55.79
0.4	8.68	25.40	38.47	46.11	51.74	56.06
0.5	9.36	27.38	38.98	46.75	52.08	57.50
0.6	10.53	28.82	38.98	46.77	46.77	58.70
0.7	12.85	30.04	38.58	47.13	53.80	59.29
0.8	11.40	29.61	38.78	46.95	53.69	59.38
0.9	9.76	28.10	38.91	46.82	52.75	58.42
1	9.04	26.03	38.76	46.40	52.30	56.63

表 5 BUTTER 和 DRETOM 在 2002 到 2009 年的召回率 (%)

		2002	2003	2004	2005	2006	2007	2008	2009
BUTTER	Top1	15.79	17.66	12.85	10.98	14.64	20.63	10.88	18.67
	Top2	26.70	29.84	30.04	17.51	34.11	31.63	34.48	41.33
	Top3	38.70	38.17	38.58	24.79	47.05	42.34	48.90	50.56
	Top4	46.41	45.38	47.13	30.24	54.90	56.01	57.55	58.00
	Top5	53.25	52.11	53.80	35.84	62.39	59.98	65.91	67.33
	Top6	61.80	58.67	59.29	40.91	68.70	62.79	70.48	69.56
DRETOM	Top1	12.22	6.12	14.83	8.18	11.44	17.72	19.81	19.67
	Top2	20.33	20.94	24.61	14.85	30.34	30.17	30.68	32.44
	Top3	27.33	30.89	31.94	22.59	44.68	37.17	37.45	52.22
	Top4	35.43	37.50	40.02	28.92	52.40	49.34	46.90	58.89
	Top5	40.77	43.87	46.88	34.91	59.85	56.27	53.64	62.67
	Top6	46.08	50.87	52.02	39.69	68.28	62.44	58.07	68.00

4.4.2 结果的评价

许多研究将缺陷分发视为分类问题, 即每个开发者为一类, 每个缺陷只能属于一类. 但是 BUTTER 算法与之不同, 它的目标是预测会在缺陷下进行评论的开发者, 而这就不局限于推荐 1 个人了. 因此, BUTTER 与传统的基于分类的缺陷分发方法没有可比性.

DRETOM 与 BUTTER 有同样的目标, 因此我们对这两种方法进行了对比. 在所有的 8 年中, BUTTER 结果都优于 DRETOM, 在某些年份中两者结果相近, 但大部分年份中 BUTTER 都远远优于 DRETOM. 从这个角度来说, BUTTER 不仅比 DRETOM 更准确, 而且更稳定. 我们认为 BUTTER 更稳定的原因是它加入了从异构网络中提取的结构信息, 有效的减少了 LDA 模型中结果太依赖于文本质量的影响.

BUTTER 算法在 Top 6 上平均召回率在 60%左右. 我们认为有以下几点原因导致了缺陷被错误地分发给不合适的开发者. 首先, 在如 Eclipse 这样的开源社区中, 开发者是自愿加入到项目中的. 通常来说, 他们可以在任意时间加入或离开. 如果一个开发者在 10 月

到 12 月之间离开, 因为这段时间内的缺陷都被视为测试集, 那么我们可能推荐这个已经离开的开发者; 相反的, 如果一个开发者在 10 月到 12 月之间加入, 因为他不在之前的训练集中, 我们不会推荐这个开发者, 但是实际上他已经在修复缺陷了. 我们不能预测开发者们加入或离开的时间, 因此这个原因导致的错误将会不可避免. 其次, 开发者有自己擅长的领域, 但是并不局限在该领域内, 他们偶尔也会由于各种原因参与其他领域缺陷的修复, 这是 BUTTER 算法所不能考虑到的情况. 尽管有以上各种错误的出现, 但我们认为 BUTTER 的结果在现阶段是令人满意的, 因为它成功的预测了大部分缺陷的真实修复者, 并且排除了数百名肯定不会参与修复的开发者, 这节省了很多人工分发缺陷的成本.

5 总结与展望

本文提出了 BUTTER 算法, 它能自动为缺陷推荐潜在的开发者, 减少人工分发缺陷的工作成本. 我们从缺陷报告中抽取其中的文本信息作为文档代表这个缺陷, 利用 LDA 对文档进行主题建模. 然后利用 RankClass 算法提取“提交者-缺陷-开发者”异构网络中的结构信息. 缺陷中包含的主题信息会通过网络传播给提交者和开发者, 因此它们能对 LDA 中产生的主题分布进行调整, 使之更加完整. 最后, 根据综合了文本信息和结构信息建立的模型, 我们可以得到更好的结果.

我们将在以下几个方面进行下一步的工作: 首先, 本文中用到的 LDA 工具和 RankClass 算法中的参数都是使用了默认值, 没有默认值的参数, 我们通过实验或查阅文献的方式自己设定. 比如主题个数 K, 我们通过一系列实验来确定效果最好的 K 取值为 10; Anvik et al.[13]认为开发者过滤的阈值是跟具体项目的领域知识相关的, 因此根据对数据的观察我们设为 90%. 在参数的选择上, 可以做更多的实验来确保其客观性和扩展性. 另外, 缺陷报告中除了已经抽取的总结、缺陷描述等文本信息和“提交者-缺陷-开发者”网络信息之外, 还包含有缺陷属于的组件、产品目录等信息, 希望以后能有效利用这些信息使模型效果更好. 还有, 我们将引入更多的数据集进行实验来验证 BUTTER 算法的可扩展性.

参考文献

1 Raymond ES. The Cathedral and the Bazaar. O’Reilly & Associates. 1999.

2 Reis CR, Fortes RP. An overview of the software engineering process and tools in the mozilla project. Proc. of the Open Source Software Development Workshop. 2002. 155–175.

3 Jeong G, Kim S, Zimmermann T. Improving bug triage with bug tossing graphs. Proc. of the 7th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering. Amsterdam, Netherlands. 2009. 111–120.

4 Cubranic D, Murphy GC. Automatic bug triage using text categorization. Proc. of the 16th International Conference on Software Engineering & Knowledge Engineering. 2004. 92–97.

5 Xuan J, Jiang H, Ren Z, Zou W. Developer prioritization in bug repositories. Proc. of the 34th International Conference on Software Engineering. 2012. 25–35.

6 Wu W, Zhang W, Yang Y, Wang Q. Drex: Developer recommendation with k-nearest-neighbor search and expertise ranking. Proc. of the 18th Asia-Pacific Software Engineering Conference. 2011. 389–396.

7 Blei DM, Ng AY, Jordan MI. Latent dirichlet allocation. The Journal of Machine Learning Research, 2003, 3: 993–1022.

8 Ji M, Han J, Danilevsky M. Ranking-based classification of heterogeneous information networks. Proc. of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining. San Diego, USA. 2011. 1298–1306.

9 Anvik J, Hiew L, Murphy GC. Who should fix this bug? Proc. of the 28th Int. Conf. on Software Engineering. Shanghai, China, 2006. 361–370.

10 Phan XH, Nguyen CT. Gibbslda++: A c/c++ implementation of latent dirichlet allocation. http://gibbslda.sourceforge.net/

11 Bhattacharya P, Neamtiu I. Fine-grained incremental learning and multi-feature tossing graphs to improve bug triaging. Proc. of the 26th IEEE International Conference on Software Maintenance. 2010. 1–10.

12 Xie X, Zhang W, Yang Y, Wang Q. Dretom: developer recommendation based on topic models for bug resolution. Proc. of the 8th International Conference on Predictive Models in Software Engineering. Lund, Sweden. 2012. 19–28.

13 Anvik J, Murphy GC. Reducing the effort of bug report triage: Recommenders for development-oriented decisions. ACM Trans. on Software Engineering and Methodology (TOSEM), 2011.

14 Heinrich G. Parameter Estimation for Text Analysis [Technical report]. University of Leipzig, Germany. 2005.