齐 齐 哈 尔 大 学

外文翻译译文正稿

题 目 利用知识转移和粗糙集进行预测通过文本挖掘了解Android测试报告的严重性

学 院 计算机与控制工程学院

专业班级 软件171

学生姓名 贾竞超

指导教师 吴迪

2021年 4月 15日

利用知识转移和粗糙集进行预测通过文本挖掘了解Android测试报告的严重性

**摘要**：众包是软件应用程序测试的一种吸引人且经济的解决方案因为它能够接触到大量的国际观众。与此同时,众包测试可能会带来很多错误报告。因此，在众包软件测试中，检验大量的测试报告是一项庞大但必不可少的软件维护任务。因此,自动预测众包测试报告的严重性非常重要，因为它们的严重性很高数字和噪音的大比例。大多数现有的解决这个问题的方法都使用了supervised机器学习技术，这通常需要用户手工标记大量的训练数据。但是，Android测试报告没有标注严重级别，也没有手工标注既费时又费力。为了解决上述问题，我们提出了一个知识基于文本挖掘和机器学习的迁移分类(KTC)方法预测测试报告的严重性。我们的方法从bug存储库和使用中获取训练数据知识转移预测Android测试报告的严重性。此外，我们的方法使用基于粗糙集的重要度约简(IDR)策略提取特征关键词到得到更准确的归一化结果。实验结果表明，本文提出的方法是可行的有利于预测android测试报告的严重性。

**关键词**：众包测试;知识转移;测试报告;重要性程度降低;粗糙集

#### 1.介绍

#### 众包技术近年来在科技领域得到了广泛的应用软件工程[1]。众包技术的一个关键优势是它们提供了工程师提供真实用户的操作信息，并由这些用户提供任务数据在真实的、多样的软件和硬件平台上执行。例如，众包测试(例如，GUI测试[2])为大量差异很大的用户提供用户体验结果，硬件、操作系统和版本。

Android漏洞追踪系统[3]是一个管理测试报告的众包测试工具从各种来源收集，包括开发团队、测试团队和最终用户被认为是众包工人。然后，Android开发团队手工分析测试报告并为每个测试报告分配优先级，以表示它对业务的紧迫性bug得到修复的视角。该测试报告的优先级是一个重要的评估，这取决于测试报告的严重程度，即缺陷对成功执行影响的严重程度软件系统。有些测试报告被标记为严重测试报告(例如，测试中的“严重”)(术语)，其相关的错误被发现是严重的问题。严格的测试报告通常有比非严重测试报告(即“非严重”)更高的修复优先级，测试报告的子集据信不会产生任何严重影响。通过这种方式，众包员工帮助了集中式员工开发人员揭露错误。然而，Android bug跟踪系统并没有保持严重程度测试报告的标签。由于众包测试产生的大量测试报告，手动标记测试报告的严重性是一项耗时且乏味的任务。因此,自动对大量测试报告的严重性进行分类的能力将非常重要促进这一过程。

之前已经进行了一些研究来调查问题报告的分类使用监督机器学习算法的开源项目[4-7]。Feng等[8,9]提出用于众包测试的测试报告优先级排序方法。他们为在每次迭代中为检查动态地选择风险最大且最多样化的测试报告。Wang等[10,11]提出了一种基于聚类的有效分类方法众包报告解决了地方偏见问题。不幸的是，Android bug报告没有用于培训数据的严重性标签，这些方法通常要求用户手工标记大量的训练数据，既费时又费力练习。因此，在仍然能够的情况下，减少手工标注的繁重负担是至关重要的达到良好的业绩。

本文提出了一种基于文本的知识转移分类方法用于预测生成的测试报告严重性的挖掘和机器学习方法众包测试。解决缺乏可用于Android测试的严重性标记训练数据的问题报告中，我们的方法从错误存储库中获取有标记的训练数据，并使用知识转移预测Android测试报告的严重性。我们应用自然语言处理技术，  
即，标记化、停止词删除和[12]词干提取，从测试报告中提取关键字。这些关键字用于预测测试报告的严重程度。尽管共识方法在实践中，不能否认关键字标签集合中仍然存在一定程度的噪声。本文采用一种基于粗糙集的重要度约简(IDR)策略进行数据提取特征关键字，减少集成标签中的噪声，从而提高培训数据和模型质量。设计了几个实验并进行了验证所提出的方法可以有效地预测Android测试报告的严重性。我们试图评估提出的KTC方法在众包环境中的性能基于对准确性，精确性和召回率的测量。

#### 2.相关工作

在软件维护活动中，问题报告是有价值的资源。自动化的支持对于问题，报告分类有助于理解、资源分配和计划。Menzies和Marcus[13]提出了一种从深坑中提取一般结论的新方法数据基于文本挖掘和机器学习的方法，具有低成本、自动化和迅速。他们设计并构建了一个名为SEVERIS(严重性问题评估)的工具来自动进行评估当提出的严重程度异常时，审查问题报告和警报。SEVRIS的构建方式提供评估正确的概率。这些可能性可以用来指导在这个过程中做决定。Tian等人[14]提出了一个名为DRONE (PreDicting通过多方面因素分析的优先级)预测Bugzilla中bug报告的优先级。他们考虑了多个因素，包括:时间、文本、作者、相关报告、严重性和产品。然后，这些特征被输入到一个名为灰色(阈值和线性)的分类引擎回归分类不平衡数据)由线性回归和阈值相结合建立解决数据不平衡问题的方法，并为bug报告分配优先级标签。Wang et al.[15]提出了一种同时使用自然语言信息和执行的方法检测重复bug报告的信息。他们的基本思想如下。首先,基于在信息检索方面，我们计算新bug报告和现有bug报告之间的两个相似点分别使用自然语言信息和执行信息的bug报告。第二,我们使用一些启发式方法使用前面的相似性来确定建议列表。Zanetti等人[16]提出了一种基于社会量化的九种度量方法对有效bug报告进行分类的方法bug报告程序嵌入到协作网络中。Zhou等人[17]提出了一种混合型结合文本挖掘和数据挖掘技术对bug报告数据进行识别纠正的方法bug报告。这样可以减少错误分类的干扰(例如，过滤错误报告)是不可纠正的)，并支持更好的bug预测性能。他们的工作基本上是一种多阶段分类方法——一种特殊的集成学习技术——组成通过一套特定的学习算法，目的是超越组成个体的人。Wang et al.[18]提出了一种将文本挖掘和数据挖掘相结合的混合方法bug报告数据自动化预测过程的技术。第一阶段利用文本挖掘技术分析bug报告的概要部分，并将它们分为三个级别的概率。然后将提取的特征和bug报告的一些其他结构化特征输入进入机器学习者的第二阶段。数据嫁接技术被用于桥梁两个阶段。Mao等人提出了一种基于内容的自动推荐技术匹配任务和开发人员。该方法从注册历史和挖掘赢家的历史，以支持适当的开发者。Yu等人使用神经网络进行预测bug报告的优先级。他们的技术还使用了类似系统的重用数据集加速进化训练阶段。Antoniol等人也将文本挖掘技术应用到对报告的bug的描述，以预测一个报告是表示真实的bug还是功能请求[19]。他们使用了决策树、逻辑回归以及Naïve贝叶斯分类器等技术为这个目的。Menzies等人基于一种规则学习技术预测bug严重程度使用了对报告的bug的文本描述。他们的方法应用于五个项目由NASA的独立验证机构提供。Hooimeijer等人建立了[20]用于识别高质量错误报告的预测模型。在他们的模型中，他们考虑错误报告当bug在给定的时间范围内是可修复的时，是高质量的。这个预测模型可以通过提取描述性信息，有效地区分高质量和低质量的bug报告  
关于错误报告、产品、操作系统和报告提交者的信息。当存储库中是否存在大量冗余的错误报告，冗余的错误报告会使其负担过重开发人员(主要是潜在的修复开发人员)，延迟项目进度，并增加项目成本。Runeson等人[21]提出了一种基于信息检索技术的冗余检测方法。此方法将每个错误报告视为一个文档，并返回一批类似于通过计算当前报告和现有报告之间的相似性来获取当前报告。Sun等人[30]提出了一种新的方法，通过构建一个判别器来检测重复的bug报告回答“两个错误报告是彼此重复的吗”这个问题的模型?该模型将报告a和B重复的概率得分。然后使用这个分数来检索相似的来自错误报告存储库的错误报告，供用户检查。我们已经研究了从包括OpenOffice在内的3个大型开源应用程序，  
Firefox和Eclipse。随后，Sun等人[22]提出了一种多特征信息检索方法可以继续匹配具有相似性的报告的最相似特征的模型检测到，从而从他们通过获取额外的信息有效地识别冗余报告。

#### 3.研究方法

在本节中，我们将介绍如何使用从错误存储库获得的知识详细预测Android漏洞的严重程度。

一份Android bug报告在其生命周期中会经历几个解决状态。当一个错误第一次报告时，错误报告被标记为未确认。当triager验证了这个bug不是重复的，实际上是新的bug，状态设置为new。然后，triager分配漏洞向一个适当的开发人员报告，并且状态被更改为Assigned。然后，分配的开发人员重现错误，对其进行本地化，并试图修复它。当错误被解决后，错误报告标记为已解析。在那之后，如果测试人员对解决方案不满意，那么这个bug应该是满意的重新开放，状态设置为重新开放;如果一个测试人员已经验证了解决方案的工作，那么状态是更改为验证。错误报告的最终状态为Closed，这是在没有发生时设置的  
bug被报告了。严重的测试报告通常比非严重的bug有更高的修复优先级报告，非严重的错误报告，认为没有任何严重影响。由于大量的bug报告，能够自动对大量的严重性进行分类测试报告将大大促进这一过程。不幸的是，Android漏洞追踪系统没有保留作为训练数据使用的严重性标签。因为有大量的测试报告在众包测试中生成，手动标记Android漏洞报告的严重性可以费时又乏味的任务。为了解决这一问题，我们对bug进行了提取和预处理从错误报告数据库生成一个训练集，它被人为地标记为严重的水平。我们还从Android bug追踪器[3]中挖掘了可用的未标记的bug报告生成一个评估集。新的错误报告被可视化为一个文本矩阵。矩阵的每一行表示一个错误报告，而矩阵的每一列表示一个单词。

下一步，我们使用提出的KTC方法来预测评估数据的严重性根据训练数据，如图1所示。然而，错误报告的数量相当高。而且，因为记者是根据自己的理解填写报告的，所以不能保证这些报告中的表达式输入正确。这可能会产生一些低质量的错误报告在数据集。此外，因为出现在文档中的每个单词都被视为附加的  
维度，文本数据可以具有非常高的维度。为了解决这两个问题，KTC该方法采用基于粗糙集的方法来减少数据中的噪声和不必要的单词降低文本数据的维数，提高预测精度。

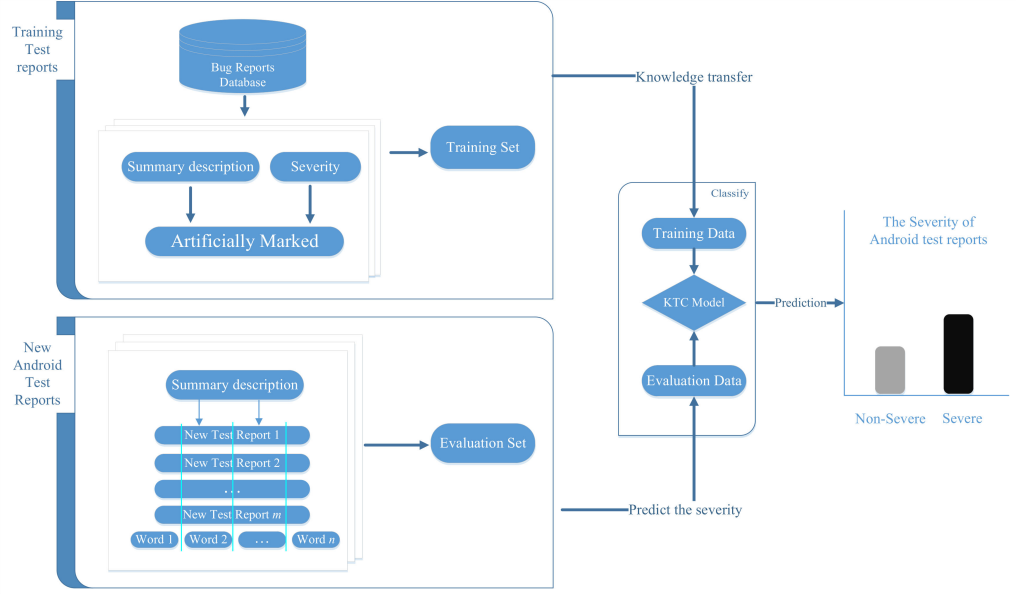


图1所示 预测测试报告严重性的框架。

3.1模型描述

在本节中，我们将详细介绍我们的KTC模型。图2显示了KTC的工作流程预测测试报告的严重性，主要包括六个步骤:(1)标记化;(2)障碍删除;(3)抑制;(4)IDR;(5)关键词向量建模;(6)分类结果。

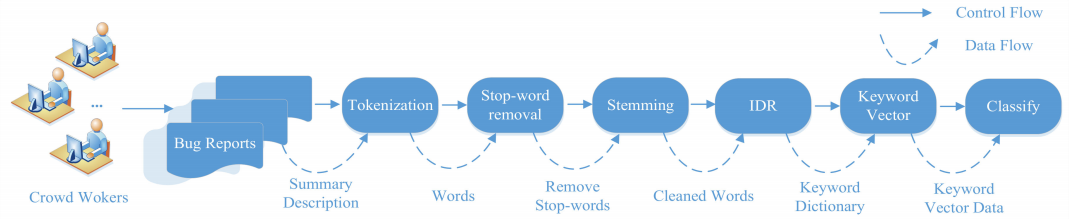


图2所示 KTC模型的工作流程，以预测测试报告的严重性。

1. 标记化:标记化过程包括将一个大的文本字符串划分为一个集合其中，单个令牌对应于单个术语。这个步骤还包括过滤排除所有无意义的符号，如标点符号，因为这些符号对分类任务。同样，所有大写字符都将被它们的小写版本所替换。
2. 移除停止词:人类语言通常使用建设性的术语，如连词，副词、介词和其他用于造句的语言结构。这样的条款正如“the”、“in”和“that”一样，它们也被称为停止词，在句子中没有什么具体的信息错误报告的上下文。此外，这些术语经常出现在bug报告描述和从而增加数据的维数，这反过来可能导致性能分类算法的减少。这有时被称为维度的诅咒。因此，根据已知的停止词列表，从标记集中删除所有停止词。
3. 词干提取:词干提取步骤的目的是减少描述中出现的每个词以最基本的形式。每一项都可以用不同的形式表达，同时仍然携带同样的具体信息。例如，术语"computerize"，"computerize"和“计算”都有相同的词源:“计算机”。这样的词干提取算法正如波特的词干，把每一个术语都转换成它的基本形式。
4. IDR:从测试报告中提取的关键词在测试预测中起着重要的作用报告严重程度。为了总结关键字中包含的信息，我们计算它们频率(即发生的次数)来估计它们的重要程度。然而,关键字是由众包工人用自然语言描述的，他们可能有各种各样的不同的背景(即专业知识、可靠性、性能、地点)。这可能会导致一个大的关键字维数和相当大的噪音。在本研究中，基于利用粗糙集提取特征关键词，降低集成标签中的噪声，从而提高了训练数据和模型的质量。IDR策略描述在下一节中详细介绍。这样，就建立了一个关键词字典。
5. 关键字向量建模:这一步涉及到构建关键字向量模型(KV)。基于关键字字典，我们为每个测试报告构造一个关键字向量，tri=(ei1,ei2，…，eim)，其中m为关键字字典中关键字的个数。我们设eij = 1如果第i份测试报告包含关键字字典中的第j个关键字且eij = 0，则不包含。
6. 分类结果:在数据中提出了多种分类算法挖掘社区。在本研究中，我们应用了四种最重要的算法(Naïve Bayes (NB)，k-最近邻(KNN)，支持向量机(SVM)，和决策树(J48))来验证我们的问题，我们的方法的准确性。

3.2 IDR策略的描述

在本节中，关键字向量的定义和关键字的重要程度是考虑到。详细说明了重要度约简算法。在我们的研究中，条件属性集C是所有关键字的集合，B (B三角形C)是实例的一个子集条件集。决策属性集D是bug报告严重性的集合。REDC (D)表示C相对于d的所有相对约简的集合。C的核心(core (C))定义为结核病∈REDD (C) B[23]。集合V表示包含关键字的频率在测试报告中。因此，将关键词约简问题转化为决策问题表减少[14]。

IDR算法见算法1。在步骤1中，我们计算相对于D的核心C。在步骤2中，如果为正区域，则令B (B三角形图C)为等于core (C)的条件集的子集在属性集B等于正决策区域的条件下，决策属性集D的在条件属性集B中，输出约简集B根据决策表的核心添加新属性。每次查找属性时，总是选择最重要度最高的属性，直到C不变。

#### 4实验设计

预测Android bug报告严重性的实验设计描述在这一节。

4.1实验装置

在实验中，我们从Android bug追踪器中随机抽取了600份Android测试报告系统[24]作为评价数据。另外，我们手工标记了严重性Android测试报告根据为bug存储库定义的规则。确保分类器不会根据缺陷的严重程度而受到缺陷分布的影响，我们为每个严重程度在训练和评估集合中选择同样多的报告。验证我们在两个主要的开源项目中使用了来自bug存储库的bug报告，即Mozilla和Eclipse，作为训练数据，如表3所示。

Android测试报告[25]:Android bug追踪系统是一个众包测试工具管理从各种来源收集的大量测试报告，包括开发团队，测试团队和最终用户，他们被认为是众包工人。在软件环境中维护时，高严重性通常对应致命错误和崩溃，而无严重性地位主要与外表问题有关。然而，Android测试报告没有严重性当前Android bug跟踪系统中的标签。

Eclipse和Mozilla bug存储库数据集是通过Bugzilla bug跟踪系统获取的，这是一个著名的bug管理平台。关于如何分配bug的严重程度。以及错误报告的严重程度，这是由专业的triager手工标记的。尽管，triager可能标记错误的严重性报告，因为错误的数量很大报告。在Bugzilla bug跟踪系统中的bug报告的生命周期之后，triager可能会改变bug的真正严重性状态是根据开发人员的评论报告的。为了保证bug报告的严重性状态是稳定的，我们选择bug报告，并在1997-2008年的时期。我们研究的错误报告来自错误存储库，其中严重性状态分为普通、次要、正常、主要、严重和阻断。Bugzilla还允许用户使用具有“严重性”增强功能的报告形式的报告报告的报告机制来请求特性。本研究不考虑这些报告，因为从技术上讲，它们并不代表真正的bug报告。在我们的方法中，我们将轻微和轻微的bug视为非严重的，而报告为major、critical和blocker严重性被认为代表严重的错误。Herraiz等人提出的[26]类似的一组严重程度。在我们的研究中，我们故意不考虑正常的严重程度。这样做的一个原因是这样的错误代表了一个灰色区域，因此可能会使分类器混淆。然而，一个更重要的原因是，在我们调查的病例中，“正常”的严重程度是在报告bug时选择严重性的默认选项，我们怀疑很多报告者只是没有费心去有意识地评估bug的严重性。错误报告的手工抽样证实了这种怀疑。

4.2评价指标

我们现在定义正确分类和错误分类的度量自然语言模型。一个真正的阳性(TP)是一个经过验证的严重测试报告，并正确分类的模型。假阳性(FP)是一个经过验证的非严重的测试报告，被错误地归类为一份严厉的测试报告。假阴性(FN)是一份经过验证的严重测试报告，但分类错误作为一份非严格的测试报告。真阴性(TN)是经过验证的正确的非严重检测报告属于非严格测试报告。

在信息检索、自然语言处理和分类问题中，精度是最重要的一个测量结果相关性的方法，而回忆是测量有多少真正相关的结果返回。因此，准确性和召回都是基于对相关性的理解和测量。这些量也与f-测度有关，f-测度被定义为精度的调和平均值和回忆。当查全率高但查准率低的结果返回很多结果时，大部分结果是预测的与训练标签相比，标签不正确。结果精度高，但精度较低Recall刚好相反，返回的结果很少，但是它的大多数预测标签都是正确的与训练标签相比。高精密度、高召回率的结果会返回很多结果，所有结果标记正确。

准确性:模型的准确性是正确分类的数量除以总数的数量分类。我们更正式地定义为:

精度:精度是指预测的错误报告的严重程度与正确程度之间的百分比预测。因此，我们分别考虑每个严重状态的精度。我们更正式地定义精度如下:

召回:召回是正确预测到的所有严重错误报告的百分比严重。在这里，我们也分别考虑每个精度的召回。更正式地说，我们定义回想一下如下:

F-measure:通常，精确度和召回分数不是单独讨论的。相反,将一种度量值与另一种度量值的固定值进行比较，或者两者都进行比较组合成一个单一的度量，如f度量，它是的加权调和平均值精度和召回。F-measure具有精度或召回率低的特点,f值也降低了。在这个案例研究中，我们定义F-measure，即召回率和准确率同样加权:

4.3 实验结果

在本节中，将讨论与具体研究问题相关的实验结果。RQ1:IDR能否提高预测android测试报告严重性的准确性?

在第一个实验中，我们比较了Eclipse中每个选定组件的精度并通过Mozilla预测android bug的严重性报告，如表4所示。在这个表中，我们用粗体突出显示最好的结果。项目列和产品+组件列显示产品和组件，我们从项目中选择来预测android bug的严重程度报告;而其他列则显示了预测android测试报告严重性的准确性四个分类器和我们的方法的四个分类器。

表4显示了使用Mozilla和Eclipse组件预测严重性的准确性android bug报告。例如，对于来自Eclipse项目的JDT产品的UI组件，NB分类预测android测试报告严重程度的准确率为0.632，准确率为采用我们的方法(IDR + NB)预测android测试报告严重程度的NB分级为0.683。此外，对Eclipse NB分类的平均准确率进行预测android的严重性测试报告为0.725，使用我们的方法(IDR + NB)对Eclipse进行NB分类的平均精度为0.725预测android测试报告的严重性为0.758。

从这些结果中，我们确实注意到对于大多数预测android漏洞严重性的人来说据报道，我们方法的分类器比标准分类器更准确。此外,平均用我们的方法对标准分类器进行预测的准确率高于标准分类器android bug报告的准确性。此外，在这个表中我们可以看到NB分类与我们的方法(IDR + NB)在使用Eclipse和Mozilla预测严重性方面具有最高的准确性android测试报告。该方法基于支持向量机和KNN的分类器具有较好的分类精度几乎与我们方法的NB分类器一样好。此外，我们看到J48分类器是一个不太准确的方法。

我们使用两个bug存储库数据集(Eclipse和Mozilla)进行知识转移，以预测严重性Android测试报告。但是，测试报告的不同表达方式采用自然语言以及众包工人的不同可靠性，这可能是造成噪音的原因预测测试报告的严重性。在这种情况下，培训的高质量bug报告的数量是相当低的，因此我们处理的是一个训练不足的分类器，自然导致可怜的准确性。为了解决这一问题，本文采用了基于粗糙集的IDR策略对提取的特征关键词进行降噪。首先，我们应用自然语言处理(NLP)技术从bug报告中获取特性关键字。我们用了很多的特性关键字，以表示bug报告和bug报告的严重程度。其次,我们使用将特征关键字构建决策信息表。最后，我们使用IDR方法决策信息表的关键字约简。我们可以去掉多余的关键词  
在不影响分类能力的前提下得到分类规则。这一结果表明我们的方法(IDR + NB)适用于使用两个bug存储库数据集(Eclipse, Mozilla)的知识转移，以预测Android测试报告的严重性。

此外，表5显示了IDR + NB的AUC值，使用Eclipse组件进行预测android测试报告的严重度分别为0.758、0.739、0.647、0.66。还有，所有人的AUCEclipse组件大约是0.701。IDR + NB的AUC值使用Mozilla组件预测android测试报告的严重程度分别为0.778、0.725、0.647、0.621。我们注意到一个改进了Mozilla组件，我们观察到AUC约为0.72。在这个情况下，这意味着我们的方法比我们随机猜测的方法的性能要好大约22%每个bug的严重程度。

因此，我们得出结论，我们的方法有效地从bug存储库获取训练数据并使用知识转移来预测Android测试报告的严重性，这是基于提供的信息，特别是使用NB分类器的单行摘要(IDR + NB)。这种方法的准确性是合理的，但这取决于具体情况。

#### 5.结论以及下一步工作

错误报告的一个关键项目是所谓的“严重性”，因此工具支持以推荐或验证系统的形式报告错误的人是可取的。在这篇文章中,提出了一种基于文本挖掘和机器的知识转移分类方法学习方法。我们的方法从错误存储库获取训练数据并使用知识转移以预测Android测试报告的严重性。从测试报告中提取关键字使用NLP技巧。因此，KV可能包含大量的关键字维而可观噪音。为了解决这一问题，我们提出了一种基于重要性度降低(IDR)策略对粗糙集上的特征关键词进行提取，获得更准确的约简结果。实验结果表明，所提出的KTC方法可以较准确地预测预测结果Android测试报告的严重性。

本文比较了四种比较知名的文档分类算法(Naïve Bayes)(NB)、k近邻(KNN)、决策树(J48)和支持向量机(SVM)来查找找出哪一种特定算法最适合将Android bug报告分类为“严重”或“非严重”类别。我们发现，对于正在调查的案件，平均准确率用分类器预测Android bug报告的严重性分别为0.715、0.716、0.721、0.659。然而，预测Android漏洞的平均准确率通过我们的分类器报告严重性方法(IDR)分别为0.735、0.731、0.734、0.663。结果表明，我们的方法(IDR)可能有助于预测Android bug报告的严重性。因此NB分类器用  
我们的方法(IDR + NB)是最适合预测Android bug报告严重性的方法。

未来的工作旨在包括更多的数据来源来支持我们的预测。来自(较长的)描述的信息将经过更彻底的预处理，以便能够使用的预测。此外，我们还将调查其他情况，其中提交的bug报告较少，但是在这里，错误报告会被有意识地审查。同时，我们可以使用主题模型来标注Android bug报告的严重性。