一种基于卷积神经网络的缺陷报告分配方法

郭世凯[1] **陈荣**[1] **唐文君**[1] **李辉**[1] **魏苗苗**[1]
(大连海事大学 辽宁省大连市 116026)[1]

摘 要 随着软件功能的越来越丰富，规模越来越大，软件缺陷不可避免的会大量出现。缺陷解决方案已经成为软件开发过程中一个重要的活动。为了减少手动分派缺陷报告的时间成本，文本分类技术被应用于自动缺陷报告分派中。本文提出了一种基于卷积神经网络的缺陷报告分派方法（BT-WCNN）。首先利用 Word2vec 方法对缺陷报告中的文本特征进行词向量表示，然后利用卷积神经网络结合批规范化，池化和全连接操作来对词向量表示的缺陷报告进行有监督的学习。我们对三个大型开源项目，即 Eclipse、Mozilla 和 NetBeans 进行了实证研究，研究结果表明 BT-WCNN 方法可以有效的提高自动缺陷报告分派的准确性。

关键词 缺陷分派，软件仓库挖缺，卷积神经网络，深度学习

中图法分类号 （细化到 3 位数字 ） **文献标识码** A **DOI** （投稿时不提供 DOI 号）

A Convolutional Neural Network Based Learning Approach for Bug Triaging

Shikai Guo[1] Rong Chen[1] Wenjun Tang[1] Hui Li[1]
(College of Information Science and Technology, Dalian Maritime University, Dalian 116026, China)[1]

Abstract Due to the unavoidable bugs appearing in the most of the software systems, bug resolution has become one of the most important activities in software maintenance. To decrease the time cost in manual work, text classification techniques are applied to automatic bug triaging. In this paper, we present a new automatic bug triaging approach which is based on convolution neural network, namely, (BT-WCNN). Firstly, the word vector representation of the text features in bug report is carried out by using Word2vec method. Then, the convolution neural network is combined with batch normalization, pooling and full connection approach to learn from the word vector representation of bug report with known fixers. We empirically investigate the accuracy of automatic bug triaging on three large open source projects, namely Eclipse, Mozilla and NetBeans. The results show that our approach can effectively improve the accuracy of automatic bug triaging.

Keywords Bug triage，Mining software repositories，Convolution neural network，Deep learning

1 引言

在软件缺陷管理流程中，缺陷报告系统（例如 Bugzilla、JIRA）扮演着重要的角色。随着软件项目的规模越来越大、功能越来越丰富，缺陷报告追踪系统每天都会接收大量的缺陷报告，据统计，修复这些缺陷将会占用软件公司 45%的开发时间[24, 25]。

当一个缺陷报告提交到缺陷报告追踪系统中之后，后台管理人员需要根据缺陷报告的描述信息来手动的将缺陷报告分派给合适的开发人员。这个为缺陷报告分派合适的开发人的过程被称为缺陷分派。然而缺陷分派是一个耗时耗力的事情，主要有两个挑战。一个是缺陷报告数量大。对于大型的项目，缺陷追踪系统每天会接收到大量的缺陷报告，仅以 Eclipse 项目为例，与其相关的缺陷报告每天可达 91 个[13]。另外一个是参与修复的开发人员数量大。对于大型项目往往需要大量的开发人员来进行开发和维护。比如 Mozilla 项目，有超过 1022 名开发人员参与到缺陷报告的修复工作中。后台管理人员很难能够熟悉每一个开发人员的擅长领域，所以如果由人来完成缺陷分派，肯定会造成准确率降低。

为了解决以上问题，缺陷报告的自动分派方法应运而生。自动缺陷报告分派通过对历史数据

到稿日期： 返修日期：
陈荣（1969-）,男，博士生，教授，主要研究方向为智能软件工程数据挖掘，E-mail: rchen@dlmu.edu.cn(通讯作者)。

进行学习统计，自动为新的缺陷报告推荐开发者。在以往的研究中，文献[6,7,8,9, 18]都是使用向量空间模型来表示缺陷报告的文本信息。文献 [5,11,19]则是使用了主题模型来分析单词在文本中的出现关系，将缺陷报告表示为在不同主题下的分布情况。另外缺陷报告中还包含其他的一些信息，比如，产品，组件和严重程度等。文献[5,11]利用这一部分信息，进一步提升了自动缺陷分派的准确率。

然而目前的自动缺陷分派方法依然存在的一些不足，主要体现在以下两个方面。首先大多研究都是将文本直接使用一位有效码（One-Hot）方法转换成向量的模式[13, 15, 16]。由于一位有效码的编码是随机的，这样会导致丢失了上下文信息；此外，由于词汇量很大，如果将所有的词汇对应的向量合为一个矩阵的话，那么这个矩阵过于稀疏且会造成维度灾难。其次，目前相关工作主要用主题模型和传统分类算法进行 bug 报告自动指派。当数据规模变大时，传统的方法解决问题时间就会变的很长。

为了避免现有方法所存在的一些问题，在本文中我们提出了一种基于卷积神经网络的缺陷报告自动分配方法。首先我们使用文本预处理方法对缺陷报告中的文本进行分词，去停词，词干化的处理。然后使用 Word2vec 方法对已经处理好的缺陷报告文本信息进行词向量表示。最后利用卷积神经网络结合批规范化，池化和全连接操作来对词向量表示的缺陷报告进行有监督的学习。为了验证我们方法的有效性，本文在 Eclipse，Mozilla，NetBeans 三个大规模数据集上进行了实验。本文总共收集了 74419 个缺陷报告，并以 top-1 到 top10 的准确度对实验结果进行评估。实验结果显示，本文的方法在 top10 的准确度达到最高，分别优于 3 个数据集中表现最好的算法 13.58%，6.34%，2.76%。

本文的主要贡献总结如下：

1. 本文提出了一种新的基于卷积神经网络的模型去解决缺陷报告分派问题。首先使用 Word2vec 方法对缺陷报告的文本信息进行词向量表示。然后使用词向量对新的卷积神经网络模型进行训练，来预测缺陷报告的开发人员。
2. 在大规模真实数据集上的实验验证了本文方法的有效性。本文在 3 个大规模、并且持续维护的开源项目的缺陷追踪系统中获取了大量的缺陷报告。在这些缺陷报告上的实验显示本文的方法性能要优于已有的方法。

2 相关研究

针对于缺陷报告自动分派的问题，研究者们提出了一系列基于机器学习或者是信息检索的方法 [5,6,9,10,12,13,18,20]。机器学习的方法通常是把缺陷报告的文本信息视为特征，将开发者视为标签，通过构造的分类器为缺陷报告自动指派开发者。而信息检索的方式则是同时对缺陷报告的文本信息和开发者进行建模，通过计算相似性来分派合适的开发人员给缺陷报告。

Anvik 等人首次提出了文本分类的方式来进行缺陷报告自动分派，并使用 Naive Bayes（朴素贝叶斯）方法来验证效果的可行性 [18]。Cubranic 等人则是在 Anvik 的基础上多尝试使用 SVM（支持向量机）、C4.8（决策树）来解决缺陷报告分派问题 [6]. Xuan[23] 针对现在缺陷报告数据中开发人员标注不准确的情况，提出了一种半监督文本分类的方法推开发者。这种方法结合了朴素贝叶斯方法和期望最大化方法。该方法首先使用小部分准确的已标注缺陷报告信息来训练朴素贝叶斯分类器，然后，迭代标注大量

未标注的缺陷报告。最后使用所有已标注的缺陷报告信息训练新的分类器，并使用加权开发人员的权重来推荐开发人员列表。Xi 等次序信息和开发者活跃度的自动缺陷分派方法 DeepTriage。不仅考虑了文本中的次序信息，还考虑了开发者的活跃程度来进行缺陷报告分派[13]。Tamrawi 等人提出了一种基于模糊集和开发者缓存的方法来进行缺陷报告分派 [9]。Xia 等人提出了 TopicMinerMTM 模型,在(Latent Dirichlet Allocation)LDA 的基础上加入了监督信息,使缺陷报告的主题受其特征（即本文所提元数据）监督,以获得更紧密的主题分布[5]。 Xuan [17]通过扩展社会网络技术分析开发人员信息对开发人员进行优先级排序。并根据开发人员排序的结果分析了 3 个影响因素，产品特性、时间变化和噪音容错。并在此基础上，研究人员根据开发者的优先级来对缺陷报告分派、严重程度预测和缺陷报告重启预测 3 方面进行了处理。Naguib 等人利用历史开发者修复信息，分析每个开发人员的经验，并使用主题模型来对比缺陷报告和开发者的相似性来进行缺陷报告分派[20]。Wu 等人使用 8 种不同的方法对开发人员的经验值进行排序，并结合使用 K 近邻搜索相似的缺陷报告进行分派 [10]。Xuan[7]针对缺陷报告分派时，由于文本信息的噪音问题而影响分派准确率问题，提出了使用数据约减方法在缺陷报告样本和单词双重维度约减得到小规模、高质量的训练集，并通过实验验证可以有效的提高缺陷报告指派的准确率。而现有的 bug 分派方法大都使用的 one-hot 词向量表示方法，该方法是文本信息中出现的每一个单词都被认为是一个维度。这样的缺点就是每个文本的有效码的编码是随机的，这样会导致丢失了上下文信息；其次 One-hot 词向量化中每个词都是相互独立的，如果将所有的词汇对应的向量合为一个矩阵的话，那么这个矩阵过于稀疏且会造成维度灾难。而本文在文本处理上使用 word2vec 方法，采用了更为合适的方法，本文考虑了文本间次序和单词之间的语义关系组成了文本信息的特征，来提供卷积神经网络学习，最终用于为 bug 报告推荐合适开发者。

在以往的工作中，还有研究人员考虑了其他的缺陷报告信息，比如，产品和组件等[8,19,25]。Yang 等人首先通过主题模型 LDA 方法来计算缺陷报告的相似性，结合多个属性信息将与当前不一致的缺陷报告过滤掉，基于这些相似的缺陷报告，推荐合适的开发者修复缺陷报告[19]。研究者们还尝试使用其他信息进行自动缺陷分派。 Jeong 等人提出了使用传递图来描述当前开发者无法修复的缺陷和缺陷报告传递给其他开发者的信息，来优化缺陷报告分派的精度[25]。Bhattacharya 等人在 Jeong 的基础上进行改进，通过分析结合多种属性的传递图来提升分派精度 [8]。而文献[22]表明，bug 报告的元信息并不可靠，只有 bug 报告中的短描述和长描述文本信息是稳定可靠的。其他所有元信息都是后台管理人员根据 bug 报告的后期修复状况进行再次修改的。因此，本文仅考虑 bug 报告的短描述和长描述来对 bug 报告进行开发人员进行指派。

3 基于卷积神经网络的缺陷分派

3.1 方法框架

根据上节的需求分析，我们提出了一种新的缺陷报告分派方法（BT-WCNN），如图 1 所示。该方法分为两个阶段：训练阶段和预测阶段。在训练阶段，通过对历史已经修复的缺陷报告进行学习并建立模型。在推荐阶段，根据新的缺陷报告信息，已训练好的模型将输出推荐的开发者列表。

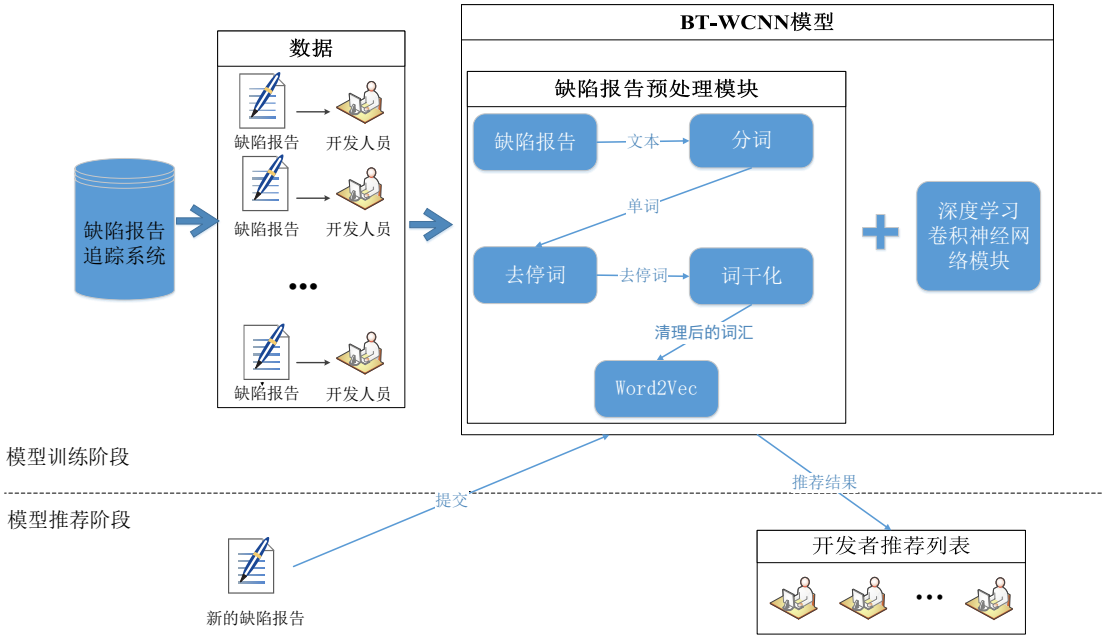

图 1　BT-WCNN 的模型框架图

Fig. 1 The Overall Workflow of BT-WCNN

在该流程中，我们首先要从原始的缺陷报告追踪系统中抽取有效的信息：文本信息（短描述，第一条长描述）和该缺陷报告的修复人员。随后这些原始信息构成了训练分类模型（BT-WCNN）的实体数据。首先 BT-WCNN 中的缺陷报告预处理模块先对文本信息进行文本预处理：分词、去停词、词干化、Word2Vec，把每一个缺陷报告都转换为一个文本矩阵。随后将这些文本矩阵训练到卷积神经网络模块中，在卷积神经网络模型中,文本矩阵信息转换为高层特征,已知的修复者为当前特征的标签通过卷积神经网络的训练会得到一个预测模型。在预测阶段,给定一个新的缺陷报告时,将其输入到模型当中便可以得到一个该缺陷报告的推荐修复者列表。

3.2　缺陷报告预处理模块

每当一个缺陷报告被提交的时候，提交人都会填写一个表单来提供缺陷的信息。然后由开发人员根据这个表单的信息来解决这个缺陷。这个表单包括对观察到缺陷的概要描述和详细描述，有时候还包含缺陷的堆栈的执行信息。对于缺陷报告中的文本信息出现每一个单词都被认为是一个额外的维度，因此，所有的缺陷报告的文本信息会是一个非常高的维度。在这里我们引入文本预处理方法(BOW)能在一定程度上克服了这个问题，减少了需要处理的单词的数量。典型的文本预处理步骤如下：

分词：分词的过程是把原始的缺陷报告的文本字符串信息转换成一组单词集合。在这一过程中还过滤掉所有无意义的符号，如标点和逗号，因为这些符号对分类任务没有贡献。此外所有的大写字符都被转换成小写字母。

去停词：缺陷报告中的文本信息通常使用诸如连词、副词、介词、和其他语言结构来组成句子。在缺陷报告的文本中，一些对指派任务无贡献的词汇会有较高的出现频率，比如：“the”、“in”和“that”。这些词汇频繁地出现会造成数据的维度灾难，从而不利于分类算法的性能提升。因此，在去停词这个过程中，删

除所有停用词。

词干化：词干化的目的是将缺陷报告中的每一个词语都简化为其最基本的形式。在自然语言中每个词语都有多种不同的表现形式。例如：“computerized”，“computerize”，“computation”都具有相同的形态基础“computer”。我们使用 Porter stemmer [14] 方法将每个单词转换为其的基本形式。

Word2Vec：为了将词干化之后的词汇输入到神经网络中,将词干化之后的词汇转换成向量的模式。一位有效码（One-Hot）是一种有效的编码方式[13, 15, 16].但是，由于一位有效码的编码是随机的，这样会导致丢失了词汇之间的信息；此外，由于词汇量很大，如果将所有的词汇对应的向量合为一个矩阵的话，那么这个矩阵会过于稀疏且会造成维度灾难。使用 Word2Vec 可以有效解决这个问题。Word2Vec 通过训练可以将每个词汇都映射到一个较短的词向量上来。一个缺陷报告是由多个词向量构成的二维矩阵，这个矩阵包含了词汇间的上下文信息，为了利用这些上下文信息，我们通过卷积神经网络来提取缺陷报告的文本特征。比如：Eclipse 编号为 212801 的缺陷报告的标题是“Creating new report fails or causes exception”,那么该缺陷报告转换成向量空间就如下面所示（Word2vec 维度设为 5）：

$$\begin{bmatrix} Creating \\ new \\ report \\ fails \\ or \\ causes \\ exception \end{bmatrix} = \begin{bmatrix} -0.0086 & 0.0205 & 0.0115 & 0.0859 & 0.0628 \\ 0.0216 & -0.0401 & -0.0611 & -0.0838 & 0.0611 \\ 0.0746 & -0.0836 & 0.0728 & -0.0014 & 0.0150 \\ -0.0518 & -0.0978 & -0.0782 & -0.0807 & -0.0230 \\ 0.0482 & 0.0948 & -0.0947 & 0.0840 & 0.0614 \\ 0.0770 & 0.0004 & 0.0197 & -0.0396 & -0.0174 \\ -0.0126 & -0.0067 & -0.0720 & 0.0725 & 0.0604 \end{bmatrix}_{7\times5}$$

3.3 卷积神经网络模块

缺陷报告分派的目标是为新来的缺陷报告自动的推荐合适的开发者。本文的方法使用的信息

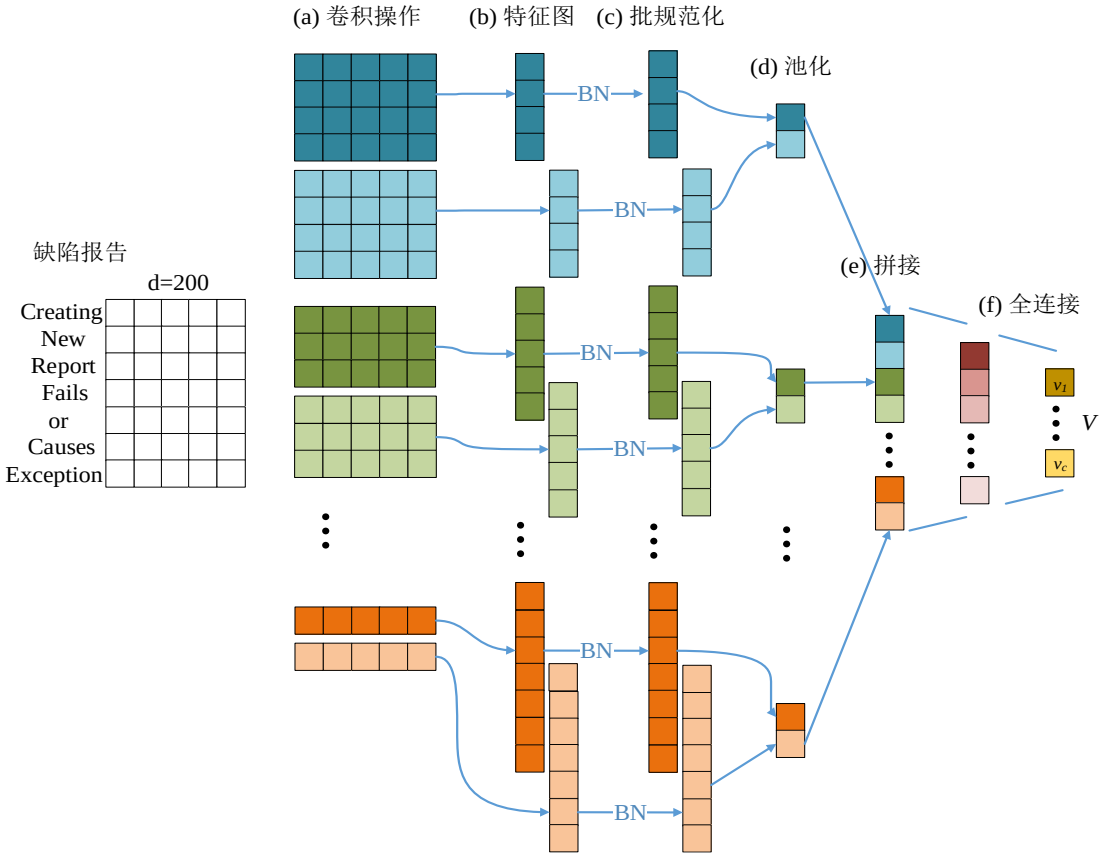

图 2　卷积神经网络模块框架图

Fig. 2 The Overall Workflow of CNN

包括文本信息和修复记录信息。文本信息经过缺陷报告预处理模块处理之后转变成矩阵表示。我们把由矩阵表示的文本信息和修复记录信息输入到卷积神经网络模块中，卷积神经网络模块对输入的文本信息、修复记录信息进行高层特征抽取，来预测合适的开发者。该模型的示意图见图 2 所示，模型包括(a)卷积核，(b)特征图，(c)批规范化，(d)池化，(e)拼接，(f)全连接 6 个操作。

以图 2 为例，首先把缺陷报告文本矩阵作为输入。为了提取不同的上下文关系，我们采用不同空间尺度的卷积核对输入的文本矩阵进行卷积操作，如图 2.(a)卷积操作所示。经过卷积过程后，可以得到具有不同语义信息的特征向量（如图 2.(b)特征图），这些特征向量包含了不同的上下文信息。接着，我们利用批量标准正态化（BN）对所有的特征向量的每一维进行处理（如图 2.(c)批规范化所示），这一步操作主要是使同一维度的数据的统计规律更加明显，从而更有利于神经网络的学习，在很多文献中，BN 对网络性能的提升作用已通过试验得以证明。BN 之后，我们采用 ReLU 做为激活函数，该函数可以避免梯度下降过程中的梯度消失问题。接着，我们使用最大池化操作（如图 2.(d)所示），通过该操作对特征向量进行降维，过滤出特征向量中较大的值，这些值构成的特征向量通过图 2.(e)拼接操作被特征融合层拼接在一起，拼接得到的特征向量具有多语义的上下文信息，并且该向量作为一个全连接层的输入（如图 2.(f)全连接操作），全连接层的输出作为一个 softmax 层的输入，该层的作用主要是将全连接层的输出结果进行归一化，使得输出向量 V，其中 V=(v_1, v_2, …, v_i, …, v_c)，c 是类别个数（我们方法把不同开发者作为类别标签），满足：v_i>0 且$\sum_{i=1}^{c} v_i = 1$，这样第 i 个输出节点就代表这个样本属于第 i 类的概率，i=1,2,…,c。因为神经网络的实际输出是一个概率向量 V=(v_1, v_2, …, v_i,…, v_c)，神经网络的期望输出也是一个概率向量 T=(t_1, t_2,…, t_i,…, t_c)，为了度量这两个向量的距离，我们采用交叉熵函数，该函数可以度量 V 和 T 之间的 Kullback-Leibler Divergence（KL 距离），该函数的函数形式如下：

$$Loss(V,T) = -\sum_{i=1}^{c} t_i \log v_i + \lambda \|W\|_2^2$$

其中，$\|\cdot\|_2^2$是 L2 正则项，λ是 L2 正则项在总体损失中的权重大小，我们在损失函数里加入 L2 正则项主要是为了防止过拟合的现象。训练神经网络的优化目标则是最小化 V和 T之间的距离，即

$$W^* = \arg\min_{W} Loss(V,T)$$

其中，W是网络的参数，这个优化过程是使用随机梯度下降方法来完成。

4 实验设计

4.1 实验数据

本文选取了 3 个规模较大，并且是持续维护的开源项目作为实验对象。项目的名称分别是：Eclipse [1]，Mozilla [2]，NetBeans [3]。所有的缺陷报告都是从其对应的缺陷追踪系统中获取到的。我们只处理缺陷报告中目前状态是 FIXED，即是 CLOSED,RESOLVED,VERIFIED 的缺陷报告。如下表 1 列出了每一个实验对象的基本数据。表格的每一列的含义如下：项目名称,缺陷报告的起止时间，收集的缺陷报告的数量，缺陷报告中的不同单词的个数，缺陷报告中不同开发者的数量。

为了进行实验，需要从每一份缺陷报告中抽

取文本信息（标题和详细描述），开发者。文本信息经过文本预处理方法进行分句、分词、词根化，去停词 [4]。此外，我们派出了修复次数少于 10 次的开发者以减少噪声[5,6,7,8]，并且我们删除了出现次数过多（超过 50%文档出现）和过少的词（少于 10 次），以减少噪声并加速模型执行速度。

表 1　缺陷报告的统计量

Table 1 Statistics of Collected Bug Reports

名称	时间区间	缺陷报告个数	单词个数	开发者个数
Eclipse	2001/10/10-2014/12/29	39669	40938	771
Mozilla	1999/03/17-2014/12/31	15501	18793	1022
NetBeans	2000/10/21-2014/12/31	19149	19451	265

参照也有的工作[5,9]处理方法，本文将被分派者（assigned to）的标签作为报告的修复者。并且与文献[5]中的观察一致，在大量的缺陷报告中发现了无效的开发者(这类开发者代表了团体的名称，或者特殊含义，这些特殊含义的开发者只对于项目的从属人员有一定的意义。在 Eclipse 中，31.15%的缺陷报告被分派给了“webmaster”，“platform-runtime-inbox”；在 Mozilla 中 13.14%的缺陷报告被分派给了“nobody”；在 NetBeans 中 14.06%的缺陷报告被分配给了“issues”。由于这些名称并非真实的开发者，因此实验剔除了这些缺陷报告。

4.2　实验设置

为了模拟现实中的场景，本文采用时间顺序的方法将数据集分割。根据缺陷报告的提出时间，前 80%作为训练集，后 20%作为测试集。实验使用准确率作为评估指标，即预测准确数量站所有测试数据的比例。实验中同时考察了 TOP-1 到 TOP10 的准确率，（TOP-k 是指真实修复的开发者是够在推荐的第 k 条之内）。

我们对比的基准算法包含 2 方面：有监督方法和无监督方法。有监督基准方法主要是由在文本分类领域表现效果好的分类器组成：朴素贝叶斯 (NB)，多项式朴素贝叶斯 (NBM)，支持向量机 (SVM)，最近邻 (KNN)，随机树 (RT)，决策树 (J48),还有席[13]提出的 DeepTriage 方法。无监督基准方法包含：DREX [10], DERTOM [12], LDA-SVM [11], LDA-KL [11].

本文的代码采用了 TensorFlow 实现。对比的基准算法普通分类器是使用 Weka [21]实现。DeepTriage 是使用 TensorFlow 实现。非监督基准方法中的主题模型 LDA 使用 Python 实现主题模型的参数设置参考原始论文。使用 TensorFlow 实现的算法采用了显卡进行加速，显卡的配置为 NVIDIA TITAN X，其他的方法皆是通过 CPU 执行，运行的 CPU 为 intel i7。

4.3　研究问题

RQ1. 使用卷积神经网络来进行缺陷报告分派是否可行？

首先,本实验关注我们的方法能否在 3 个实验对象上胜过一些基准的有监督方法。本文考虑如下 7 个基准有监督对比方法：在传统文本分类取得比较好效果的 6 种分类算法：NB,NBM,SVM,KNN,RT,J48 和 Xi [13] 提出的 DeepTriage 方法，利用双向循环网络加池化的方法来提取特征报告的文本特征，另一方面利用单向循环网络提取特定时刻的开发者活跃度特征，结合两者来对缺陷报告进行有监督的指派。

RQ2. 与传统有监督方法相比，无监督方法性能如何？

和传统方法对比，有监督学习往往效率很低，因此有很多的研究人员使用无监督方法进行

缺陷分派。Wu [10]提出了 DREX 方法，该方法基于结合 K 近邻求解缺陷报告的相似性和开发者经验排序的方法来进行缺陷报告分派。Xie [12]提出了 DERTOM，使用主题模型来对缺陷报告分派开发人员。Somasundaram [11]提出了 LDA_SVM 和 LDA_KL 方法，该方法是分别结合支持向量机和相对熵来对缺陷报告分派开发人员。

5 实验结果

RQ1. 使用卷积神经网络来进行缺陷报告分派是否可行？

在第一个研究问题中，我们把 BT-WCNN 和传统的有监督基准方法进行对比（NB, NBM, SVM, KNN, RT,J48, DeepTriage），来验证 BT-WCNN 方法对缺陷报告进行分派知否是可行的。表 2-4 展示了 BT-WCNN 和 7 种有监督的基准方法的 top-1 到 top10 的准确率。表 2-4 中，对于每一行的最大值表示该方法获得最高的准确率，我们用加粗显示。比如，在表 2 中 BT-WCNN 在 Top10 情况下获取了 0.5337 的准确率，是其他 7 种有监督方法中最高的，我们把 0.5337 加粗。

从表 2-4 中，我们可以发现 BT-WCNN 方法的准确率相对于其他 7 种基准方法有的一定的提高。7 种基准算法中，NBM 方法获得了最高准确率。在表 2 的 Eclipse 项目中，我们可以发现 BT-WCNN 分别比 NB, NBM, SVM, KNN, RT, J48, DeepTriage 方法在 Top10 上提升了 92.99%，21.8%，99.90%，92.99%，63.71%，25.44%.在表 3 的 Mozilla 项目中，我们可以发现 BT-WCNN 分别比 NB, NBM, SVM, KNN, RT, J48, DeepTriage 方法在 Top10 上提升了 31.24%，16.2%，88.47%，80.49%，81.90%，50.56%，18.20%。在表 4 的 NetBeans 项目中，我们可以发现 BT-WCNN 分别比 NB, NBM, SVM, KNN, RT, J48, DeepTriage 方法在 Top10 上提升了 27.90%，23.1%，66.52%，70.60%，70.56%，45.43%，2.76%。

表 2-4 中，我们 BT-WCNN 方法在 NetBeans 项目进行缺陷报告分派预测获取的准确率最高，Mozilla 进行缺陷报告分派预测获得的准确率最低。我们通过手动分析数据，发现 Mozilla 项目的人员流动较大，新加入的开发者较多，而 BT-WCNN 的分类方法是在模型训练好之后，就不能预测新加入的开发者，对于其他的有监督方法也有的同样的问题。

从表 2-4 中，我们可以得到 BT-WCNN 方法在对比有监督的基准方法在缺陷报告分派预测上，可以获得更高的准确率。但是也有很多学者使用无监督方法对缺陷报告分派进行预测，为了全面验证 BT-WCNN 方法的性能，在研究问题 2 中，我们将 BT-WCNN 方法和几种常见的无监督方法进行对比。

表 2　对于 Eclipse 项目，同有监督基准方法对比 top1-top10 的准确率

Table 2 Top-1 to top-10 accuracies for BT-WCNN with baseline approaches for Eclipse

Eclipse	BT-WCNN	BOW+NB	BOW+NBM	BOW+SVM	BOW+KNN	BOW+RT	BOW+J48	DeepTriage
Top-1	**0.2520**	0.0374	0.1029	0.0006	0.0396	0.0374	0.1144	0.1247
Top-2	**0.3508**	0.0374	0.2428	0.0006	0.0396	0.0374	0.1489	0.1959
Top-3	**0.3970**	0.0374	0.2973	0.0006	0.0396	0.0374	0.1573	0.2657
Top-4	**0.4328**	0.0374	0.3295	0.0006	0.0396	0.0374	0.1865	0.2887
Top-5	**0.4603**	0.0374	0.3519	0.0006	0.0396	0.0374	0.1883	0.3160
Top-6	**0.4834**	0.0374	0.3704	0.0006	0.0396	0.0374	0.1885	0.3376
Top-7	**0.4988**	0.0374	0.3836	0.0006	0.0396	0.0374	0.1900	0.3567
Top-8	**0.5124**	0.0374	0.3981	0.0006	0.0396	0.0374	0.1937	0.3743
Top-9	**0.5236**	0.0374	0.4086	0.0006	0.0396	0.0374	0.1937	0.3877
Top-10	**0.5337**	0.0374	0.4174	0.0006	0.0396	0.0374	0.1937	0.3980

表 3　对于 Mozilla 项目，同有监督基准方法对比 top1-top10 的准确率

Table 3 Top-1 to top-10 accuracies for BT-WCNN with baseline approaches for Mozilla

Mozilla	BT-WCNN	BOW+NB	BOW+NBM	BOW+SVM	BOW+KNN	BOW+RT	BOW+J48	DeepTriage
Top-1	**0.1244**	0.0815	0.0906	0.0226	0.0509	0.0460	0.0997	0.0826
Top-2	**0.1909**	0.1143	0.1463	0.0226	0.0509	0.0460	0.1334	0.1383
Top-3	**0.2254**	0.1369	0.1833	0.0352	0.0627	0.0582	0.1477	0.1728
Top-4	**0.2491**	0.1578	0.2045	0.0352	0.0634	0.0582	0.1547	0.1986
Top-5	**0.2693**	0.1749	0.2233	0.0387	0.0662	0.0613	0.1599	0.2251
Top-6	**0.2826**	0.1868	0.2404	0.0397	0.0672	0.0624	0.1624	0.2411
Top-7	**0.3017**	0.1976	0.2551	0.0397	0.0672	0.0624	0.1652	0.2547
Top-8	**0.3171**	0.2129	0.2669	0.0397	0.0672	0.0624	0.1693	0.2648
Top-9	**0.3300**	0.2265	0.2780	0.0397	0.0672	0.0624	0.1704	0.2749
Top-10	**0.3446**	0.2369	0.2889	0.0397	0.0672	0.0624	0.1704	0.2819

表 4　对于 NetBeans 项目，同有监督基准方法对比 top1-top10 的准确率

Table 4 Top-1 to top-10 accuracies for BT-WCNN with baseline approaches for NetBeans

NetBeans	BT-WCNN	BOW+NB	BOW+NBM	BOW+SVM	BOW+KNN	BOW+RT	BOW+J48	DeepTriage
Top-1	**0.3554**	0.2157	0.2917	0.1478	0.1184	0.1184	0.2163	0.3484
Top-2	**0.4802**	0.3041	0.3972	0.1478	0.1184	0.1184	0.2783	0.4742
Top-3	**0.5436**	0.3649	0.4311	0.1478	0.1184	0.1184	0.2979	0.5387
Top-4	**0.5893**	0.4039	0.4612	0.1548	0.1251	0.1251	0.3069	0.5747
Top-5	**0.6266**	0.4359	0.4847	0.1548	0.1254	0.1257	0.3170	0.6157
Top-6	**0.6539**	0.4572	0.5091	0.1585	0.1302	0.1307	0.3212	0.6461
Top-7	**0.6811**	0.4825	0.5273	0.2508	0.2202	0.2205	0.3621	0.6733
Top-8	**0.7111**	0.5018	0.5447	0.2508	0.2202	0.2205	0.3933	0.7069
Top-9	**0.7273**	0.5195	0.5627	0.2508	0.2202	0.2205	0.4084	0.7112
Top-10	**0.7489**	0.5400	0.5759	0.2508	0.2202	0.2205	0.4087	0.7283

RQ2. 与传统有监督方法相比，无监督方法性能如何？

在第二个研究问题中，我们把 BT-WCNN 方法和 4 种无监督基准方法进行对比（DREX, LDA-SVM, LDA-KL, DERTOM）,来验证我们 BT-WCNN 方法的性能。表 5-7 展示了 BT-WCNN 方法和 DREX, LDA-SVM, LDA-KL, DERTOM 方法的 top1 到 top10 的准确率。类似表 2-4，对于每一行的最大值表示该方法获得最高的准确率，我们用加粗显示。

从表 5-7 中，我们可以发现 BT-WCNN 方法的准确率相对于 DREX, LDA-SVM, LDA-KL, DERTOM 方法有的一定的提高。在表 5 的 Eclipse 项目中，我们可以发现 BT-WCNN 分别比 DREX, LDA-SVM, LDA-KL, DERTOM 方法在 Top10 上提升了 35.6%，13.58%，21.35%，23.10%。 在表 6 的 Mozilla 中，我们可以发现 BT-WCNN 分别比 DREX, LDA-SVM, LDA-KL, DERTOM 方法在 Top10 上提升了 19.16%，6.46%，6.34%，10.26%. 在表 6 的 NetBeans 项目中，我们可以发现 BT-WCNN 分别比 DREX, LDA-SVM, LDA-KL, DERTOM 方法在 Top10 上提升了 25%，14.59%，40.97%，29.67%. 表 2-7 中，我们发现使用无监督方法对于缺陷报告分派的平均准确率，NetBeans 项目要优于 Eclipse 项目，Eclipse 项目要优于 Mozilla 项目。此外，对比有监督方法，无监督方法对缺陷报告的分派的平均准确率要高于有监督方法。这是因为对于项目数据集中有新来的开发人员，无监督方法可以对于新加入的开发者非常敏感，而有监督方法是在训练模型固定之后，无法预测新加入的开发者。

RQ3. 使用不同的词向量表示方法，对结果性能影响如何？

在这个实验中，我们对比使用不同词向量表示方法来分析对结果的影响。在这个实验中我们分别使用 word2vec 方法结合 CNN(BT-WCNN)和 one-hot 词向量方法结合 CNN。表 8-10 展示了 BT-WCNN 方法和 one-hot+CNN 方法的 top1 到 top10 的准确率。表 8-10 中，对于每一行的最大值表

表 5　对于 Eclipse 项目，同无监督基准方法对比 top1-top10 的准确率

Table 5 Top-1 to top-10 accuracies for BT-WCNN with baseline approaches for Eclipse

Eclipse	BT-WCNN	DREX	LDA_SVM	LDA_KL	DERTOM
Top-1	**0.2520**	0.1366	0.1469	0.2054	0.1234
Top-2	**0.3508**	0.1988	0.2582	0.2516	0.2179
Top-3	**0.3970**	0.2421	0.3090	0.2894	0.2762
Top-4	**0.4328**	0.2797	0.3450	0.3246	0.3129
Top-5	**0.4603**	0.3070	0.3769	0.3514	0.3424
Top-6	**0.4834**	0.3365	0.4044	0.3763	0.3681
Top-7	**0.4988**	0.3563	0.4266	0.3928	0.3895
Top-8	**0.5124**	0.3699	0.4409	0.4082	0.4046
Top-9	**0.5236**	0.3824	0.4543	0.4273	0.4209
Top-10	**0.5337**	0.3935	0.4698	0.4398	0.4335

表 6 对于 Mozilla 项目，同无监督基准方法对比 top1-top10 的准确率

Table 6 Top-1 to top-10 accuracies for BT-WCNN with baseline approaches for Mozilla

Mozilla	BT-WCNN	DREX	LDA_SVM	LDA_KL	DERTOM
Top-1	**0.1244**	0.0878	0.0854	0.0673	0.0718
Top-2	**0.1909**	0.1526	0.1380	0.1261	0.1289
Top-3	**0.2254**	0.1822	0.1857	0.1714	0.1693
Top-4	**0.2491**	0.2028	0.2188	0.2059	0.2004
Top-5	**0.2693**	0.2240	0.2436	0.2324	0.2289
Top-6	**0.2826**	0.2380	0.2634	0.2544	0.2470
Top-7	**0.3017**	0.2491	0.2822	0.2721	0.2676
Top-8	**0.3171**	0.2606	0.2965	0.2916	0.2847
Top-9	**0.3300**	0.2732	0.3091	0.3077	0.3031
Top-10	**0.3446**	0.2892	0.3237	0.3240	0.3125

表 7　对于 NetBeans 项目，同无监督基准方法对比 top1-top10 的准确率

Table 7 Top-1 to top-10 accuracies for BT-WCNN with baseline approaches for NetBeans

NetBeans	BT-WCNN	DREX	LDA_SVM	LDA_KL	DERTOM
Top-1	**0.3554**	0.2729	0.2945	0.1672	0.2396
Top-2	**0.4802**	0.3675	0.4000	0.2438	0.3529
Top-3	**0.5436**	0.4348	0.4701	0.3150	0.4065
Top-4	**0.5893**	0.4724	0.5215	0.3630	0.4429
Top-5	**0.6266**	0.5083	0.5540	0.4059	0.4710
Top-6	**0.6539**	0.5360	0.5818	0.4415	0.5032
Top-7	**0.6811**	0.5537	0.6031	0.4668	0.5206
Top-8	**0.7111**	0.5736	0.6241	0.4929	0.5380
Top-9	**0.7273**	0.5879	0.6379	0.5091	0.5574
Top-10	**0.7489**	0.5992	0.6536	0.5313	0.5776

示该方法获得最高的准确率，我们用加粗显示。

从表 8-10 中，我们可以发现，BT-WCNN 方法要优于 Onehot+CNN 方法。在 Eclipse, Mozilla, NetBeans 项目 top10 的准确率上 BT-WCNN 方法要分别优于 Onehot+CNN 方法 24.29%，4.87%，8.29%。因为 one-hot 词向量方法中，每个单词在词向量里面的位置是随机的，这样会导致丢失了上下文信息，并且相比于 word2vec 方法使用多个数据集进行训练，one-hot 词向量化方法只能针对当前数据集进行词向量化，所包含的矩阵文本信息量相比于 word2vec 较小。所以相较于 BT-WCNN 方法，使用 One-hot+CNN 方法，卷积神经网络从使用 one-hot 表述的词向量矩阵中，获取的信息量更少。从实验结果中也验证了这个问题，使用 BT-WCNN 方法的性能要比 One-hot+CNN 方法更好。

表 8 使用不同词向量方法对比 top1-top10 的准确率

Table 8 Top-1 to top-10 accuracies for different word vector method with baseline approaches

Eclipse	BT-WCNN	One-hot+CNN
Top-1	**0.2520**	0.1973
Top-2	**0.3508**	0.2949
Top-3	**0.3970**	0.3128
Top-4	**0.4328**	0.3394
Top-5	**0.4603**	0.3536
Top-6	**0.4834**	0.3619
Top-7	**0.4988**	0.3867
Top-8	**0.5124**	0.3918
Top-9	**0.5236**	0.4193

Top-10	**0.5337**	0.4294

表 9 使用不同词向量方法对比 top1-top10 的准确率

Table 9 Top-1 to top-10 accuracies for different word vector method with baseline approaches

Mozilla	BT-WCNN	One-hot+CNN
Top-1	**0.1244**	0.0816
Top-2	**0.1909**	0.1569
Top-3	**0.2254**	0.1967
Top-4	**0.2491**	0.2076
Top-5	**0.2693**	0.2357
Top-6	**0.2826**	0.2519
Top-7	**0.3017**	0.2718
Top-8	**0.3171**	0.2835
Top-9	**0.3300**	0.3011
Top-10	**0.3446**	0.3286

表 10 使用不同词向量方法对比 top1-top10 的准确率

Table 10 Top-1 to top-10 accuracies for different word vector method with baseline approaches

NetBeans	BT-WCNN	One-hot+CNN
Top-1	**0.3554**	0.3049
Top-2	**0.4802**	0.4462
Top-3	**0.5436**	0.4876
Top-4	**0.5893**	0.5391
Top-5	**0.6266**	0.5618
Top-6	**0.6539**	0.6029
Top-7	**0.6811**	0.6379
Top-8	**0.7111**	0.6661
Top-9	**0.7273**	0.6793
Top-10	**0.7489**	0.6916

6 评估的有效性威胁

1. 内部的有效性威胁

在实验设置中，训练集和测试集是根据时间排序进行划分的。这样的划分方法是和软件项目的开发进行是有区别的。对于某一阶段的软件项目，开发人员和软件的功能是相对比较固定的，但是随着软件版本的更新，项目组的人员和项目的专有名词也会变化。若是在测试集中将这部分信息考虑进来，开发者和词汇的变化会严重影响模型的性能。

2. 外部的有效性威胁

实验只对 3 个开源数据集（Eclipse，Mozilla，GNOME）进行结果分析。在未来的研究中，我们需要在更多的软件项目缺陷报告中进行实验，以便能够更好的验证我们算法的性能。

7 未来工作和总结

在本文中我们提出了一种基于卷积神经网络的自动缺陷报告分派方法。首先我们使用文本预处理方法对缺陷报告中的文本进行分词，去停词，词干化的处理。然后使用 Word2vec 方法对已经处理好的缺陷报告文本信息进行词向量表示。最后将缺陷报告的词向量信息输入到我们提出的新的卷积神经网络模型中，预测缺陷报告分派的开发人员。为了验证我们方法的有效性，本文在 Eclipse，Mozilla，NetBeans 三个大规模数据集上进行了实验。本文总共收集了 74419 个缺陷报告，并以 top-1 到 top10 的准确度对实验结果进行评估。在未来工作中，我们要应用 BT-WCNN 方法到更多的开源缺陷报告数据集中。此外我们将考虑改进新加入开发者 BT-WCNN 方法不够敏感的问题。

致　谢 国家自然科学基金（NO.61672122, NO.61602077, NO. 61771087）、辽宁省科学事业公益研究基金（No. 20170005)、辽宁省自然科学基金（NO.20170540097）、中央高校基本科研业务费资助项目（NO.3132016348）和大连海事大学优秀科技创新团队培育计划资助项目（NO.3132016348）资助课题。

参考文献

[1] Eclipse 缺陷数据库, https://bugs.eclipse.org/bugs/, Last access: 2018-08.

[2] Mozilla 缺陷数据库, https://bugzilla.mozilla.org/, Last access: 2018-08.

[3] Netbeans 缺陷数据库, https://netbeans.org/bugzilla/, Last access: 2018-08.

[4] Tao Zhang, Jiachi Chen, Geunseok Yang, Byungjeong Lee, Xiapu Luo, “Towards more accurate severity prediction and fixer recommendation of software bugs”, in *Journal of Systems and Software 117*, 2016, pp.166-184.

[5] Xin Xia, David Lo, Ying Ding, Jafar M. Al-Kofahi, Tien N. Nguyen and Xinyu Wang, “Improving Automated Bug Triaging with Specialized Topic Model”, in *IEEE Transactions on Software Engineering.43(3)*, 2017, pp.272-297.

[6] John Anvik, Lyndon Hiew and Gail C. Murphy, “Who should fix this bug?”, in *International Conference on Software Engineering*, 2006, pp.361-370.

[7] Jifeng Xuan, He Jiang, Yan Hu, Zhilei Ren, Weiqin Zou, Zhongxuan Luo, Xindong Wu. Towards Effective Bug Triage with Software Data Reduction Techniques. *IEEE Transactions on Knowledge and Data Engineering*, vol.27, no.1, Jan. 1 2015, pp.264-280.

[8] Pamela Bhattacharya, Iulian Neamtiu, “Fine-grained incremental learning and multi-feature tossing graphs to improve bug triaging”, in *IEEE International Conference on Software Maintenance*, 2010, pp.1-10.

[9] Tamrawi A, Nguyen T T, Al-Kofahi J M, et al. “Fuzzy set and cache-based approach for bug triaging”, in *Sigsoft/fse'11, ACM Sigsoft Symposium on the Foundations of Software Engineering*, 2011, pp.365-375.

[10] Wenjin Wu, Wen Zhang, Ye Yang, Qing Wang, “DREX: Developer Recommendation with K-Nearest-Neighbor Search and Expertise Ranking”, in *APSEC*, 2011, pp.389-396

[11] Kalyanasundaram Somasundaram, Gail C. Murphy, “Automatic categorization of bug reports using latent Dirichlet allocation”, in *ISEC*, 2012, pp.125-130.

[12] Xihao Xie, Wen Zhang, Ye Yang, Qing Wang, “DRETOM: developer recommendation based on topic models for bug resolution”, in *PROMISE*, 2012, pp.19-28.

[13]席圣渠，“DeepTriage:一种基于循环神经网络的缺陷报告分派方法”，in 软件学报 2018.29(8), 2018, pp.2322-2335.

[14] M. Porter, “An algorithm for suffix stripping,” in *Program 14(3)*, 1980, pp.130－137.

[15] Johnson R, Zhang T, “Supervised and semi-supervised text categorization using LSTM for region embeddings”, in *ICML*, 2016, pp.526-534.

[16] Yoon Kim, “Convolutional Neural Networks for Sentence Classification”,in *EMNLP*, 2014, PP.1746-1751.

[17] Jifeng Xuan, He Jiang, Zhilei Ren, Weiqin Zou. Developer Prioritization in Bug Repositories. *Proceedings of 34th International Conference on Software Engineering* (ICSE 2012), Zurich, Switzerland. June 2-9, 2012, pp. 25-35.

[18] Davor Cubranic, Gail C. Murphy, “Automatic bug triage using text categorization”, in *SEKE*, 2004, pp.92-97.

[19] Yang G, Zhang T, Lee B, “Towards Semi-automatic Bug Triage and Severity Prediction Based on Topic Model and Multi-feature of Bug Reports”, in *Computer Software and Applications Conference*, 2014, pp.97-106.

[20] Naguib H, Narayan N, Brügge B, et al. “Bug report assignee recommendation using activity profiles” in *Mining Software Repositories*, 2013, pp.22-30.

[21] Weka, https://www.cs.waikato.ac.nz/ml/weka/, Last access:2018-08.

[22] Ahmed Lamkanfi, Serge Demeyer, Quinten David Soetens, Tim Verdonck, “Comparing Mining Algorithms for Predicting the Severity of a Reported Bug”, in *CSMR*, 2011, pp. 249-25.

[23] Jifeng Xuan, He Jiang, Zhilei Ren, Jun Yan, Zhongxuan Luo. Automatic Bug Triage using Semi-Supervised Text Classification. *Proceedings of 22nd International Conference on Software Engineering and Knowledge Engineering* (SEKE 2010), Redwood City, California, USA. July 1-3, 2010, pp. 209-214.

[24] Sepp Hochreiter, Jürgen Schmidhuber, “Long short-term memory”, in Neural

computation 9(8), 1997, pp.1735-1780.

[25] Gaeul Jeong, Sunghun Kim, Thomas Zimmermann, “Improving bug triage with bug tossing graphs”, in Joint Meeting of the European Software Engineering Conference and the ACM Sigsoft Symposium on the Foundations of Software Engineering, 2009, pp.111-120.