DeepTriage：一种基于深度学习的软件缺陷自动分配方法

宋化志，马于涛

（武汉大学 计算机学院，湖北 武汉 430072）

E-mail: ytma@whu.edu.cn

摘 要：在软件开发和维护过程中，缺陷修复工作有一项必不可少的任务，那就是缺陷分配。在大规模的软件项目中，基于文本分类的自动分配技术已被用于提高缺陷分配的效率，从而减少人工分配的等待时间和成本。考虑到缺陷报告文本内容的复杂性，本文提出了一种基于深度学习的缺陷自动分配方法，在词向量化后通过卷积神经网络对缺陷报告文本进行特征提取，然后完成分类任务。在 Eclipse 和 Mozilla 两个数据集上的结果表明，与传统的支持向量机和基于递归神经网络的方法相比，文本所提方法在准确率指标上均优于上述基准方法，而且多层平行的卷积神经网络结构比单层的卷积神经网络结构在预测效果上更好。

关键词：缺陷分配；深度学习；卷积神经网络；递归神经网络；支持向量机

中图分类号：TP311.5 **文献标识码：**A **文章编号：**1000-1220（2009）02--

DeepTriage: An Automatic Triage Method for Software Bugs Using Deep Learning

SONG Hua-zhi, MA Yu-tao

（State Key Laboratory of Software Engineering, Wuhan University, Wuhan 430072, China）

Abstract: Triaging software bugs is an essential task of bug repair during the process of software development and maintenance. In a few large-scale software projects, automated bug triage techniques based on text categorization have been used to reduce the time and cost of manual allocation. Considering the complexity of the contents of bug reports, we propose an automatic triage method for bugs using deep learning, called DeepTriage. By using a convolutional neural network (CNN), DeepTriage extracts text features of each bug report represented by word embedding and then predict possible software developers for a given bug. The experimental results on the Eclipse and Mozilla datasets show that Deeptriage performs better than the traditional support vector machine and recurrent neural networks in term of accuracy. In particular, the multi-level parallel CNN outperforms the single-layer one.

Key words: bug triage; deep learning; convolutional neural network; recurrent neural network; support vector machine

1 引言

软件维护的目标不仅是为了修复软件缺陷，更是为了改善软件特性从而提升软件质量。随着软件开发的复杂度越来越高，软件维护也面临着巨大挑战。为了提高软件维护的效率，缺陷跟踪系统 BTS（Bug Tracking System）被广泛用于缺陷管理[1,2]，如 Bugzilla 和 JIRA。在 BTS 中，每个缺陷被描述成具有规范格式的缺陷报告，包含缺陷重复出现的描述信息以及缺陷的状态更新信息[3]等，但形式各异且存在大量冗余。开源项目中新的缺陷报告每天不断涌入并累积。例如，Eclipse 在 2007 年平均每天要新增 30 个缺陷报告[4]；从 2001 年到 2010 年，接近 3 万 5 千名开发者或用户提交了超过 33 万个缺陷报告[5]。使用人工的方法来处理类似大规模的缺陷报告是一项极具挑战性的任务。

缺陷分配是将一个新的缺陷报告分配给一个最合适的开发者并尽快修复的过程[4,6-8]。新缺陷报告的分配工作往往是由专家级的开发者来完成的。但是，这种人工分配的方式不仅耗间，而且效率也不高。例如，在人工分配的 Eclipse 缺陷报告中，有 44%的缺陷报告分配错误，而且从一个新的缺陷报告形成到第一次分配给开发者，平均要花费 19.3 天[4]。因此，为了减少人工分配的人力成本和时间成本，Anvik 等人[6]较早提出了缺陷自动分配的方法。他们将每个缺陷报告映射为一个文档，将缺陷的修复者映射为这篇文档的标签，从而使用文本分类的技术来预测合适的开发人者。另一方面，从开发者合作关系的角度 Jeong 等人[4]较早提出了基于投掷图（tossing graph）的方法。他们的实验表明，考虑开发者之间的投掷信息有助于进一步提高预测效果并缩短缺陷报告投掷的路径长度。Bhattacharya 等人[9]的工作和后续的相关研究[10,11]也表明，这类综合文本分类和投掷图的混合方法确实能获得更好的预测效果。

目前，深度学习技术已成功应用于图像分类、语音识别、文本翻译等领域，其优势在于通过组合低层特征形成更抽象的高层特征，以发现数据的分布式特征表示。考虑到缺陷报告中文本的多样性和复杂性，本文的基本思路是将缺陷报告映射为一个二维的向量矩阵，利用深度学习算法挖掘隐藏在

收稿日期：2009-10-1 基金项目：国家重点基础研究发展计划（No. 2014CB340404）资助；国家自然科学基金（Nos. 61672387, 61702378）资助；武汉市黄鹤英才（现代服务）计划资助。 作者简介：**宋化志，**男，1992 年生，硕士研究生，研究方向为软件工程；**马于涛，**男，1980 年生，博士，副教授，中国计算机学会高级会员，研究方向为软件工程、服务计算。

文本内容中的隐式特征，并完成分类任务。遵循该思路，本文提出了一种基于卷积神经网络 CNN（Convolutional Neural Network）的缺陷自动分配方法，其主要贡献如下：

（1）针对缺陷报告中的文本信息，经过词向量化（word embedding）处理后，利用 CNN 对文本特征进行自动提取，然后构造相应的分类器。在 Eclipse 和 Mozilla 上的实验结果表明，所提方法在准确率（accuracy）方面优于以往研究中表现最好的支持向量机 SVM（Support Vector Machine）。

（2）使用不同的深度学习算法进行对比实验，发现多层平行的 CNN 结构在 Eclipse 和 Mozilla 两个数据集上都能取得最好的预测效果，是一种有效的人工神经网络结构。

本文余下部分包括：第 2 节介绍和本文相关的研究工作，第 3 节介绍相关的背景知识，第 4 节详细描述实验模型以及实验步骤，第 5 节分析和讨论实验结果，第 6 节总结全文并对今后工作进行展望。

2 相关工作

2.1 基于文本分类的缺陷分配

Cubranic 等人[12]较早将缺陷分配问题转化为文本分类问题，利用朴素贝叶斯 NB（Naive Bayes）算法来预测新的缺陷报告应该分配给谁来解决最为合适。由于仅仅使用了 Eclipse 缺陷报告的描述（description）和总结（summary）信息，准确率只有 30%。Anvik 等人[6,7]扩展了上述工作，使用 NB、SVM、C4.5 决策树等机器学习算法进行对比，发现 SVM 的准确率最高。Ahsan 等人[13]使用特征选择和隐语义索引 LSI（Latent Semantic Indexing）方法来降低文档输入矩阵的维度，然后使用 SVM 取得了最佳的预测效果，平均精确率（precision）和召回率（recall）分别达到 30%和 28%。上述方法对文本的处理大多采用词袋模型，但词袋模型往往忽略重要的词序信息。在本文中，我们使用的卷积神经网络（CNN）可以弥补这一不足，即通过设置不同的卷积核大小可以捕捉不同的短语，从而提高文本挖掘的效率。

Xie 等人[10]利用斯坦福大学的主题建模工具 TXT（Stanford Topic Modeling Toolbox）将缺陷报告按照主题进行分组，然后通过分析开发人员在不同分组的兴趣和经验，为新的缺陷报告推荐一组合适的开发者。实验表明这种方法优于 SVM、KNN（k-Nearest Neighbor）等机器学习算法。类似地，Yang 等人[14]也使用 TMT 工具来确定新的缺陷报告的主题，然后利用多特征和社交网络技术来预测合适的开发者。Naguib 等人[15]将概率生成主题模型 LDA（Latent Dirichlet Allocation）引入缺陷分配中，并提出了相应的排序算法来预测合适的开发者。他们的实验表明，该方法的平均准确率可达到 88%。Zhang 等人[11]也使用 LDA 模型来挖掘历史缺陷报告的主题，然后提取具有相同主题的开发人员之间的相互关系，并从中抽取相关特征来对开发者进行排序预测。与文献[10][15]所提的方法相比，其 F-measure 值分别提高了 3.3%和 16.5%。此外，Xuan 等人[16]提出了一种半监督学习的文本分类方法——基于期望最大化的 NB 方法，其结果的准确率比单纯的 NB 方法提高了 6%。上述使用主题模型进行建模的方法也存在一些限制，比如主题数目的设定、主题建模粒度的限定等。与此相对应的是，本文中我们提出的方法没有这样的限制。

2.2 基于深度学习的文本分类

Collobert 等人[17]使用卷积过滤器处理连续文本来获取局部特征，并利用最大池化（max-pooling）获取全局特征，在不需要大量先验知识的情况下，在标记系统（tagging system）中取得了良好的效果。Kim 等人[18]使用一种包含多种过滤器和两种词向量通道的单层 CNN，在情感分析和问题分类中取得了不错的效果。Kalchbrenner 等人[19]提出了基于 k-max-pooling 机制的动态 CNN 方法，在情感预测和问题分类任务中比基准方法的错误率下降了 25%。进一步地，Lei 等人[20]在对单词进行处理时使用基于张量代数的操作来代替标准卷积层中词向量连接的线性运算，以获取非连续词向量的非线性关系，并在标准的情感分析任务中取了当时最好的结果。Zhang 等人[21]也提出基于字符的 CNN 模型，其结果优于传统的词袋（bag of words）模型和基于单词的 CNN 模型。

Zhou 等人[22]综合 CNN 和递归神经网络 RNN（Recurrent Neural Network）各自的优点，提出了 C-LSTM 分类模型，在情感分类和问题分类任务中，其效果优于单独的 CNN 和 LSTM（Long Short-Term Memory）模型。Johnson 等人[23]提出了基于 LSTM 的区域向量（region embedding）模型，充分利用半监督的神经网络方法和 two-view 词向量方法的优势，在给定数据集上均取得了当时最好的结果。Miyato 等人[24]将对抗和虚拟对抗训练拓展到文本领域，对 RNN 中的词向量而非原始输入进行扰动，在多个基准的半监督和监督任务中取得了很好的效果。

总的来说，虽然深度学习在文本挖掘和文本分类领域研究势头强劲，但在软件缺陷自动分配领域，还暂时缺少将深度学习技术引入到缺陷自动分配中的工作，相关的技术、方法和实施流程值得深入研究。

3 方法框架

3.1 词向量模型

在自然语言处理中，文本常常被转化为向量，以便计算机进行处理。向量化文本的粒度一般分为字符、单词、短语等，考虑到缺陷报告中文本的实际情况，本文采用的是广泛使用的词向量化。单词粒度的文本向量化方法一般又可以分为两类表示方法：稀疏向量和稠密向量。稀疏向量采用一位有效编码（*one-hot*）向量形式，即每个向量的维度为整个词汇表的大小，只有单词在相应词汇表位置的数字为 1，其余全为 0。稠密向量，其向量长度一般为 100~600 维，远远小于词汇表的大小，但它可以表示词与词之间更丰富的语义相似程度。因此，本文采用的是稠密向量表示方法。

一般来说，要想得到一组好的稠密词向量，不仅需要好的表示模型和参数调试，更需要大量的语料数据。本文使用提前训练好的词向量作为基准向量，即基于 Word2Vec[25]的 CBoW（Continuous Bag-of-Words）模型在 Google News 数据集（约 1000 亿个单词）上训练的词向量模型（下载网址：https://code.google.com/archive/p/word2vec）。

3.2 卷积神经网络

一个 CNN 由两部分组成：卷积和池化。卷积用于捕捉局部特征，不同的卷积核捕捉大小不一的局部特征；池化用于将卷积层捕捉的特征按一定方式提取并加以利用，如求最大值、求均值等。本文 CNN 模型的设计和构建过程如下：

首先，将每个缺陷报告的英文文本提取出来，经过分词、去停用词、提取词干、词向量化等一系列文本预处理后，得到相对“规范”的文本。

其次，假设一个文本的长度为 s，其中每个单词都可使用 Word2Vec 训练的 d 维向量表示（或采用随机数进行向量表示），那么该文本可表示为一个 $s\times d$ 维的向量矩阵。与 Collobert 等人[26]的工作类似，本文使用的卷积核的宽度与表示词向量的维度相同。因为在词向量的所有维度上，每个维度都是相互独立的，只需在行维度进行卷积操作即可。此外，假设卷积核的长度为 h，并用其代表卷积的区域大小。

再次，假设使用 $\boldsymbol{W}$ 来表示区域大小为 h 的卷积核的权重矩阵，卷积核的参数量为 $h\times d$。这里，缺陷报告文本的向量矩阵记为 $\boldsymbol{A}\in\mathbb{R}^{s\times d}$（$\mathbb{R}^n$表示 n 维的坐标空间），其子矩阵记为 $A[i:j]$，表示矩阵 $\boldsymbol{A}$ 从第 i 行到第 j 行的子矩阵。每次卷积操作可得到一个输出序列（向量），记为 $\boldsymbol{o}^t\in\mathbb{R}^{s-h+1}$，

其中的元素为：

$$o_i^t = \boldsymbol{W} \cdot A[i:i+h-1], \quad (1)$$

其中，$i \in [1, s\text{-}h+1]$，运算符“·”表示矩阵之间的数量积。如果对输出序列 $\boldsymbol{o}^t$ 加上偏置 $\boldsymbol{b} \in \mathbb{R}^{s-h+1}$ 和激活函数 f，那将得到一个特征映射（feature map）序列 $\boldsymbol{c}^t \in \mathbb{R}^{s-h+1}$，其中的元素为：

$$c_i^t = f(o_i^t + b_i). \quad (2)$$

然后，由于每个特征映射向量的维度是由文本的长度和区域大小共同决定的，本文用池化操作将每个特征映射转化为一个固定大小的向量。一种常见的池化策略是 *1-max* 池化[26]，它是从每个特征映射序列 $\boldsymbol{c}^t$ 中提取一个具有最大值的标量，这里记为 s_i。利用该池化操作，将从特征映射中提取出来的结果进行连接便组成了“最高层特征”，记为 $\boldsymbol{S} = [s_1, s_2, \cdots, s_m] \in \mathbb{R}^m$，然后将其放入 *Softmax* 层便可得到分类结果，记为 $\tilde{y} \in \mathbb{R}^k$。本文缺陷修复的类别标签使用 *one-hot* 形式表示（共有 k 维），假设 $\boldsymbol{V} \in \mathbb{R}^{k \times m}$ 表示 *Softmax* 层（*softmax*()函数）的权重矩阵，分类的类别可由公式(3)计算得到。最后，将最小化分类的交叉信息熵（如公式(4)所示）作为训练目标（即损失函数），利用给定的训练集训练预测模型。

$$\tilde{y} = softmax(\boldsymbol{V} \cdot \boldsymbol{S}). \quad (3)$$

$$L(y, \tilde{y}) = -\sum_{i=1}^{k} y_i \log \tilde{y}_i. \quad (4)$$

在 *Softmax* 层本文使用 *Dropout*[27]作为一种正则化方法，此外也可采用 L2 正则化来防止过拟合的发生。这里，需要估计的参数包括卷积核的权重矩阵、激活函数里的偏置以及 *softmax*()函数的权重矩阵。网络的优化方法采用的是 SGD（Stochastic Gradient Descent）和反向传播算法[28]。由于篇幅限制，本文省略了具体的推导过程。

算法 1：单层卷积神经网络 S_CNN 的预测算法（Python）

输入：训练文本集 Train_Text，测试文本集 Test_Text，训练文本集标签 Train_Label

输出：测试文本集标签 Test_Label

```
BEGIN
    TRAIN
        # 生成 batch 序列对
        {(Batch_Xs, Batch_Ys)} = Batches_Generator(
                                    Train_Text, Train_Label)
        FOR Batch_X, Batch_Y IN { (Batch_Xs,
                                    Batch_Ys) }:
            X = Embedding(Batch_X) # Embedding 层
            # Relu 为激活函数
            Conv_X = Relu(W1 · X + B1) # 卷积层
            Pool_X = Max_Pool(Conv_X) # 池化层
            # 将池化后的特征连接起来
            Feature_X = Concatenate(Pool_X)
            P = Softmax(W2 · Feature_X + B2) # 公式(3)
            Cost = Sum(- Batch_Y · Log P) # 公式(4)
            使用 Adam 优化器最小化目标 Cost
            并更新变量：W1, B1, W2, B2
        END FOR
    END TRAIN
    TEST
        Test_Label = [] # 初试标签列表
        {(Batch_Xs)} = Batches_Generator(Test_Text)
        FOR Batch_X IN {(Batch_Xs)}:
            X = Embedding(Batch_X)
            Conv_X = Relu(W1 · X + B1)
            Pool_X = Max_Pool(Conv_X)
            Feature_X = Concatenate(Pool_X)
            P = Softmax(W2 · Feature_X + B2)
            Y_Pred = ArgMax(P) # 得到概率最大的标签
            Test_Label.Append(Y_Pred)
        END FOR
    END TEST
    RETURN Test_Label
END
```

值得一提的是，本文设计了两种不同的卷积结构来进行实验。第一种，单层 CNN（S_CNN）采用相同的卷积区域长度，并使用 100 个卷积核进行训练，其结构如图 1 所示，相应的预测算法描述（伪码使用 Python 语言编写）见算法 1；第二种，多层平行 CNN（M_CNN）采用不同的卷积区域长度，并使用平行结构进行训练，其结构如图 2 所示，相应的预测算法描述与 S_CNN 基本一致，不同的是在算法中将其 09 和 10 步骤并行处理。

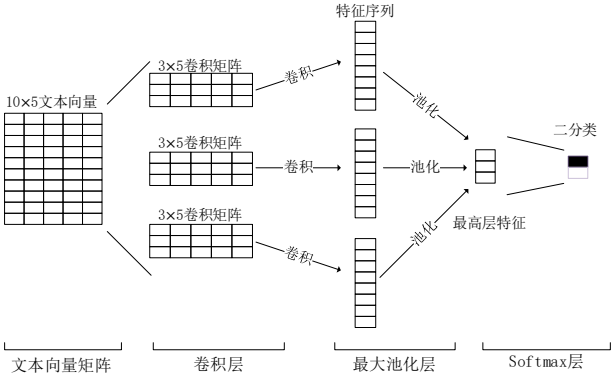

图 1 S_CNN 的网络结构

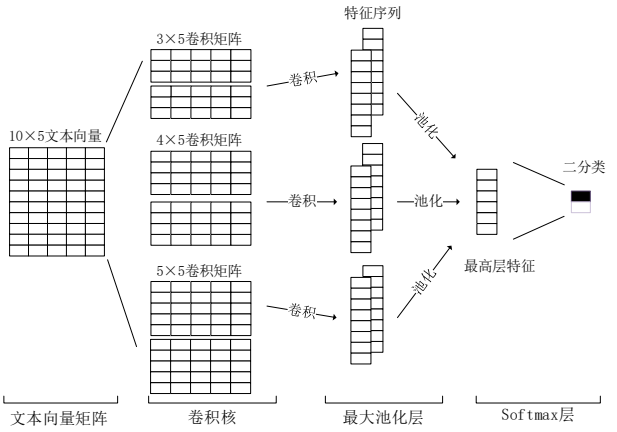

图 2 M_CNN 的网络结构

4　实验及结果分析

4.1　数据准备与预处理

在软件开发过程中，一旦发现软件出现缺陷需要修复，相关人员会以报告的形式将其提交到缺陷库中形成缺陷报告。一个缺陷报告主要包括三个部分：第一部分是总结和预定义字段（主要包括状态（status）、所属产品（product）、所属组件（component）等）；第二部分是描述，介绍如何实现缺陷的复现，帮助开发者了解缺陷发生的过程，以便查找缺陷产生的原因；第三部分是评论（comment），主要记录开发者针对缺陷的一些思考和建议。

本文选取了 Bugzilla 中 Eclipse 和 Mozilla 两个大型开源软件项目的缺陷数据作为实验的数据集，原始数据源于文献[29]。以 Eclipse 为例介绍数据的收集过程。本文选取了 2001.10~2011.09 这 10 年间产生的 20 万个缺陷报告，它们的状态为 VERIFIED、决议（resolution）为 FIXED。每个缺陷报告对应的总结、描述、评论等文本信息放在一起，组成一个文本，然后进行自然语言处理，主要包括分词、去除非字母表字符、去除停用词、词干提取、词向量化等操作。对于缺陷报告的样本类别，本文将最后一个解决缺陷的开发者作为最终的类别标签，而不是考虑所有在缺陷投掷过程中出现的开发者以及发表过评论的开发者。同样地，本文也收集了 1998.04~2011.05 这 13 年间的 22 万个 Mozilla 缺陷报告。实验数据集的基本情况如表 1 所示。

表 1 实验数据集

软件项目	缺陷编号	时间	数据量
Eclipse	1~357573	2001.10~2011.0	20 万
Mozilla	37~660036	1998.04~2011.05	22 万

4.2 实验设计

4.2.1 实验环境

本文实验是在 Dell T5810 Precision 图形工作站进行的，其配置为 8 核 Intel E5-1620V3 处理器、16GB 内存、单卡图形处理器 GPU（型号为华硕 GTX 1080）。图形工作站的操作系统为 Ubuntu 16.04，实验所采用的深度学习框架为 TensorFlow 1.1.0，Python 版本为 3.5.3。实验所用算法的具体配置见网址 https://github.com/huazhisong/graduate_text。

4.2.2 验证模式

为了模拟实际的应用场景，按照文献[30-33]中的“十折”增量（ten-fold incremental）方式进行验证。将按时间先后顺序排序的数据集等分为互不相交的 11 份。首先，在第 1 轮时，使用第 1 份数据作为训练集，第 2 份数据作为测试集；在第 2 轮时，使用第 1 份和第 2 份数据作为训练集，第 3 份数据作为测试集；依此类推，到第 10 轮时，以第 1 到第 10 份数据作为训练集，第 11 份数据作为测试集。这样，针对每种算法，我们都可以在不同的训练集上训练十次。最后，以 10 轮结果的平均值作为最终的结果。

4.2.3 基准方法

SVM 一般用于做二分类应用（当然，SVM 也可以改造为支持多分类的模型），它的基本模型是定义在特征空间上的间隔最大的线性分类器。SVM 的学习策略就是间隔最大化，可形式化为一个求解凸二次规划的问题，也等价于正则化的合页损失函数的最小化问题。间隔最大使 SVM 有别于感知机（perceptron）。除了作为线性分类器，SVM 还包括核技巧，这使它也成为了一种非线性分类器。值得一提的是，之前的工作（如文献[6,7,13]）已证明了 SVM 在缺陷自动分配问题上的高效性和优势。

RNN 是一种可以通过链式结构来存储和传递历史信息的神经网络结构。当处理序列数据时，它的输入由当前输入值 x_t 和前一个节点的隐藏节点的输出 h_{t-1} 组成。虽然 RNN 已成功应用于自然语言处理的多个应用，但由于有时连续多个输入之间的间隔过大，导致标准的 RNN 很难学习到长期的依赖，而且训练不容易收敛。Hochreiter 等人[34]提出了一种新的结构 LSTM，而且 Ilya 等人[35]将其应用到机器翻译领域，取得了良好的效果。这里，本文选择文献[34]提出的标准 LSTM 模型作为对比实验的基准模型；此外，也选取了文献[36]提出的 BiLSTM 作为对比实验的基准模型。

4.2.4 评价指标

和文献[4,12,29,33]一样，本文采用的评价指标为准确率（也称为命中率），其计算公式如下：

$$Accuracy_k = \frac{T_k}{N}, \tag{5}$$

其中，T_k 为在给定 k 名开发者的情况下预测成功（与类别标签完全匹配）的缺陷报告数目，N 代表缺陷报告的总数。

4.3 实验结果

表 2 和表 3 显示的是本文所提方法和基准方法在 Eclipse 数据集上分别预测 1 名和 5 名开发者的结果。总的来说，本文提出的 CNN 模型要明显优于基准模型。在仅预测 1 名开发者的情况下，S_CNN 的平均准确率比 SVM 的增加了 15.69%，而 M_CNN 增加了 22.96%。在预测 5 名开发者的情况下，与 SVM 相比，上述两种模型的平均准确率也分别增加了 10.87%和 17.48%。类似地，表 4 和表 5 显示了相关所有方法在 Mozilla 数据集上的预测效果。在预测 1 名和 5 名开发者的情况下，M_CNN 的平均准确率比 SVM 的分别增加了 16.01%和 14.50%。这表明 CNN 在基于文本分类的软件缺陷自动分配问题上是有效的。

相比较 CNN 而言，RNN 模型在 Eclipse 数据集上似乎并不十分有效，其效果远低于 CNN 模型的。甚至与 SVM 相比，LSTM 模型取得的效果也并不占优，而 BiLSTM 的效果要更差一点，这说明需要更好地挖掘和利用文本的序列信息，来进一步改善 RNN 的预测效果。

表 2 Eclipse 数据集中 Accuracy@1 的实验结果（%）

Accuracy	1	2	3	4	5	6	7	8	9	10	AVG
SVM	30.69	38.72	39.53	40.71	41.29	40.45	42.74	43.62	44.70	40.27	40.27
S_CNN	41.85	51.76	55.19	55.13	62.16	54.84	62.88	61.87	61.69	52.29	55.96
M_CNN	50.27	57.96	62.08	61.69	67.19	66.59	68.90	68.42	69.15	60.02	**63.23**
LSTM	35.59	35.74	35.97	41.53	36.53	42.62	42.91	48.71	44.79	28.48	39.29
BiLSTM	35.95	36.17	35.13	39.79	37.31	40.90	42.74	32.14	37.59	31.99	36.97

表 3 Eclipse 数据集中 Accuracy@5 的实验结果（%）

Accuracy	1	2	3	4	5	6	7	8	9	10	AVG
SVM	48.43	50.23	52.26	54.33	56.32	58.34	61.24	48.43	50.23	52.26	57.05
S_CNN	51.62	61.64	66.57	66.98	75.16	69.35	75.40	74.28	73.92	64.26	67.92
M_CNN	60.70	67.40	73.30	72.40	78.90	79.21	80.87	80.38	79.92	72.27	**74.53**
LSTM	50.57	53.26	54.69	59.26	55.42	61.19	61.37	66.35	61.61	44.60	56.83
BiLSTM	50.32	53.38	52.76	57.05	55.82	58.43	59.06	50.18	53.80	46.75	53.75

表 4 Mozilla 数据集中 Accuracy@1 的实验结果（%）

Accuracy	1	2	3	4	5	6	7	8	9	10	AVG
SVM	35.41	36.9	37.08	40.58	41.71	40.98	42.5	42.99	44.63	42.65	35.41

S_CNN	19.40	33.04	32.69	38.63	50.88	50.26	60.65	55.66	57.86	36.39	43.54
M_CNN	29.43	47.82	44.64	50.01	56.21	53.88	64.21	63.73	58.65	45.66	**51.42**
LSTM	17.45	25.05	27.60	24.46	24.85	35.94	41.57	37.78	46.20	39.71	32.06
BiLSTM	16.84	24.09	25.12	23.62	33.77	22.85	42.71	31.76	37.11	28.54	28.64

表 5 Mozilla 数据集中 Accuracy@5 的实验结果（%）

Accuracy	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**AVG**
SVM	43.21	44.23	46.55	48.44	49.46	50	51.24	54.25	55.32	54.12	49.68
S_CNN	30.45	45.52	41.88	49.54	63.12	63.43	73.51	66.85	69.25	49.64	55.32
M_CNN	42.645	60.33	54.49	60.66	68.015	68.525	79.065	76.16	71.98	59.925	**64.18**
LSTM	32.04	41.02	40.97	37.01	38.82	51.80	59.25	58.34	61.98	55.27	47.65
BiLSTM	29.49	39.16	38.27	35.96	51.56	36.82	61.49	47.55	53.27	42.71	43.63

在 Mozilla 数据集上，RNN 模型的预测效果也比本文提出的 CNN 模型的效果要差，而且差距比较明显，如表 4 和表 5 所示。类似地，从平均准确率指标来讲，LSTM 模型接近于 SVM，但是 BiLSTM 模型要逊于 SVM。

4.4 高效性的理论分析

本文提出的基于 CNN 的缺陷自动分派算法比基准的机器学习 SVM 算法、深度学习 LSTM 算法和 BiLSTM 算法都要高效，原因分析如下：

首先，在这些方法中，CNN 的运行效率最高。CNN 本身的结构是平行的，非常适合并行计算，加上使用了 GPU 加速，因而在运行效率上要比 SVM 快，更比 LSTM 这种串行结构的算法要快。

其次，CNN 比 SVM 更注重词序性。SVM 以加速 N-Gram 形式来增加词序信息，但是这样不仅加大了数据的维度，增加了计算量，而且会增加其矩阵的稀疏性，反而有可能降低分类的准确性。值得注意的是，LSTM 是适合用来处理序列信息的，但是由于实验数据的文本长度均接近 200 个英文单词，这么长的序列在很大程度上限制了 LSTM 读取文本序列的能力，因为它要处理的依赖过长。

最后，CNN 能快速有效地捕获局部词序信息。CNN 虽然在文本建模上的效果比不上 LSTM，但是 CNN 能捕获局部的词序信息，并将其全局共享。这一特性对文本分类效果的影响尤为重要，因为关键的文本分类信息往往就是一两个区域的短语决定的。

4.5 效度分析

虽然本文提出的基于 CNN 的文本分类方法在缺陷自动分配中取得了良好的效果，但是该方法的局限性（仅仅只考虑文本特征，未涉及缺陷所属项目和组件、开发者活跃程度等其他因素）以及一些潜在的威胁（potential threats）可能会影响实验的结果。

首先，文本直接将缺陷报告文本进行向量化表示，然后使用 CNN 自动获取文本特征，在这个过程中并未使用自然语言处理领域最新的方法，如生成对抗网络（Generative Adversarial Network）和注意力模型（Attention Model）。这表明该方法还存在进一步提升的空间。

其次，由于受硬件设备的条件限制，本文中 CNN 的网络结构是比较简单的（浅层）神经网络结构，这显然不能达到当前最好的分类水平。因此，后续工作将继续设计更深层的神经网络结构进行实验。

然后，基准方法只选取了 SVM、LSTM 和 BiLSTM 三种经典的分类算法/模型，而且其参数均使用的默认设置。由于一些复杂的机器学习分类算法和最近提出的人工神经网络模型缺乏实现代码，难以复现其效果，本文并没有考虑将它们作为基准方法进行对比。

最后，本文的实验只在 Eclipse 和 Mozilla 两个数据集上进行了测试，因而并不能保证本文所提方法在其他开源软件项目上的通用性。

5 总结

针对软件缺陷报告文本的特点，本文将深度学习中的 CNN 模型引入到软件工程领域的缺陷自动分配任务中。利用缺陷报告的总结、描述和评论等文本信息训练开发者预测模型，在 Eclipse 和 Mozilla 两大数据集中进行了实验，本文所提方法与机器学习的 SVM 算法和深度学习的 RNN 模型相比，在准确率上具有明显优势，从而验证了 CNN 结构的有效性，为缺陷自动分配问题提供了一种新的解决思路。

今后的研究工作主要包括：1）引入最新的生成对抗网络和注意力模型等方法提升文本特征的提取效果，从而进一步提高 CNN 模型的预测效果；2）考虑缺陷所属项目和组件、开发者活跃程度等因素，构建针对异构特征的混合模型。

References:

[1] Fitzgerald B. The transformation of open source software [J]. Management Information Systems Quarterly, 2006, 30(3): 587-598.

[2] Breu S, Premraj R, Sillito J, et al. Information needs in bug reports: improving cooperation between developers and users [C]. Proceedings of the 2010 ACM Conference on Computer Supported Cooperative Work (CSCW'10), 2010: 301-310.

[3] Bettenburg N, Just S, Schroter A, et al. What makes a good bug report? [C]. Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of Software Engineering (FES'08), 2008: 308-318.

[4] Jeong G, Kim S, Zimmermann T, et al. Improving bug triage with bug tossing graphs [C]. Proceedings of the 7th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT International Symposium on Foundations of Software Engineering (ESEC/FSE'09), 2009: 111-120.

[5] Xuan J, Jiang H, Ren Z, et al. Developer prioritization in bug repositories [C]. Proceedings of the 34th International Conference on Software Engineering (ICSE'12), 2012: 25-35.

[6] Anvik J, Hiew L, Murphy G C, et al. Who should fix this bug? [C]. Proceedings of the 28th International Conference on Software Engineering (ICSE'06), 2006: 361-370.

[7] Anvik J, Murphy G C. Reducing the effort of bug report triage: Recommenders for development-oriented decisions [J]. ACM Transactions on Software Engineering and Methodology, 2011, 20(3): 10:1-10:35.

[8] Park J, Lee M, Kim J, et al. COSTRIAGE: a cost-aware triage algorithm for bug reporting systems [C]. Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence (AAAI'11), 2011: 139-144.

[9] Bhattacharya P, Neamtiu I, Shelton C R, et al. Automated,

highly-accurate, bug assignment using machine learning and tossing graphs [J]. Journal of Systems and Software, 2012, 85(10): 2275-2292.
[10] Xie X, Zhang W, Yang Y, et al. DRETOM: developer recommendation based on topic models for bug resolution [C]. Proceedings of the 8th International Conference on Predictive Models in Software Engineering (PROMISE'12), 2012: 19-28.
[11] Zhang T, Yang G, Lee B, et al. A Novel Developer Ranking Algorithm for Automatic Bug Triage Using Topic Model and Developer Relations [C]. Proceedings of the 21st Asia-Pacific Software Engineering Conference (APSEC'14), 2014, 1: 223-230.
[12] Cubranic D, Murphy G C. Automatic bug triage using text categorization [C]. Proceedings of the Sixteenth International Conference on Software Engineering & Knowledge Engineering (SEKE'04), 2004: 92-97.
[13] Ahsan S N, Ferzund J, Wotawa F, et al. Automatic Software Bug Triage System (BTS) Based on Latent Semantic Indexing and Support Vector Machine [C]. Proceedings of the Fourth International Conference on Software Engineering Advances (ICSEA'09), 2009: 216-221.
[14] Yang G, Zhang T, Lee B, et al. Towards Semi-automatic Bug Triage and Severity Prediction Based on Topic Model and Multi-feature of Bug Reports [C]. Proceedings of the IEEE 38th Annual Computer Software and Applications Conference (COMPSAC'14), 2014: 97-106.
[15] Naguib H, Narayan N, Brugge B, et al. Bug report assignee recommendation using activity profiles [C]. Proceedings of the 10th Working Conference on Mining Software Repositories (MSR'13), 2013: 22-30.
[16] Xuan J, Jiang H, Ren Z, et al. Automatic Bug Triage using Semi-Supervised Text Classification [C]. Proceedings of the 22nd International Conference on Software Engineering & Knowledge Engineering (SEKE'10), 2010: 209-214.
[17] Collobert R, Weston J, Bottou L, et al. Natural Language Processing (Almost) from Scratch [J]. Journal of Machine Learning Research, 2011, 12(Aug): 2493-2537.
[18] Kim Y. Convolutional Neural Networks for Sentence Classification [C]. Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP'14), 2014: 1746-1751.
[19] Kalchbrenner N, Grefenstette E, Blunsom P, et al. A Convolutional Neural Network for Modelling Sentences [C]. Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (ACL'14), 2014, 1: 655-665.
[20] Lei T, Barzilay R, Jaakkola T S, et al. Molding CNNs for text: non-linear, non-consecutive convolutions [C]. Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing (EMNLP'15), 2015: 1565-1575.
[21] Zhang X, Zhao J J, Lecun Y, et al. Character-level convolutional networks for text classification [C]. Proceedings of the 29th Annual Conference on Neural Information Processing Systems (NIPS'15), 2015: 649-657.
[22] Zhou C, Sun C, Liu Z, et al. A C-LSTM Neural Network for Text Classification [J]. ArXiv Preprint, ArXiv:1511.08630, 2015.
[23] Johnson R, Zhang T. Supervised and semi-supervised text categorization using LSTM for region embeddings [C]. Proceedings of the 33rd International Conference on Machine Learning (ICML'16), 2016: 526-534.
[24] Miyato T, Dai A M, Goodfellow I J, et al. Virtual Adversarial Training for Semi-Supervised Text Classification [J]. ArXiv Preprint, ArXiv:1605.07725v1, 2016.
[25] Mikolov T, Sutskever I, Chen K, et al. Distributed Representations of Words and Phrases and their Compositionality [C]. Proceedings of the 27th Annual Conference on Neural Information Processing Systems (NIPS'13), 2013: 3111-3119.
[26] Collobert R, Weston J. A unified architecture for natural language processing: deep neural networks with multitask learning [C]. Proceedings of the Twenty-Fifth International Conference on Machine Learning (ICML'08), 2008: 160-167.
[27] Hinton G E, Srivastava N, Krizhevsky A, et al. Improving neural networks by preventing co-adaptation of feature detectors [J]. ArXiv Preprint, ArXiv:1207.0580, 2012.
[28] Rumelhart D E, Hinton G E, Williams R J, et al. Learning representations by back-propagating errors [J]. Nature, 1988, 323(6088): 533-536.
[29] Liu H, Ma Y. Developer Recommendation Method for Automatic Software Bug Triage [J]. Journal of Chinese Computer Systems, 2017, 38(12): 2747-2753. (in Chinese)
[30] Xia X, Lo D, Wang X, et al. Accurate developer recommendation for bug resolution [C]. Proceedings of the 20th Working Conference on Reverse Engineering (WCRE'13), 2013: 72-81.
[31] Tamrawi A, Nguyen T T, Alkofahi J M, et al. Fuzzy set and cache-based approach for bug triaging [C]. Proceedings of the 19th ACM SIGSOFT Symposium on the Foundations of Software Engineering (FSE'11), 2011: 365-375.
[32] Bhattacharya P, Neamtiu I. Fine-grained incremental learning and multi-feature tossing graphs to improve bug triaging [C]. Proceedings of the 26th IEEE International Conference on Software Maintenance (ICSM'10), 2010: 1-10.
[33] Bettenburg N, Premraj R, Zimmermann T, et al. Duplicate bug reports considered harmful … really? [C]. Proceedings of the 24th IEEE International Conference on Software Maintenance (ICSM'08), 2008: 337-345.
[34] Hochreiter S, Schmidhuber J. Long Short-Term Memory [J]. Neural Computation, 1997, 9(8): 1735-1780.
[35] Sutskever I, Vinyals O, Le Q V, et al. Sequence to Sequence Learning with Neural Networks [C]. Proceedings of the 28th Annual Conference on Neural Information Processing Systems (NIPS'14), 2014: 3104-3112.
[36] Zhang S, Zheng D, Hu X, et al. Bidirectional Long Short-Term Memory Networks for Relation Classification [C]. Proceedings of the 29th Pacific Asia Conference on Language, Information and Computation (PACLIC'15), 2015: 73-78.

附中文参考文献：

[29] 刘海洋,马于涛. 一种针对软件缺陷自动分派的开发者推荐方法[J]. 小型微型计算机系统, 2017, 38(12): 2747-2753.