软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software, [doi: 10.13328/j.cnki.jos.000000]

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

基于深度学习的安全缺陷报告预测方法实证研究*

郑 炜[1,5,6], 陈军正[1], 吴潇雪[2], 陈 翔[1,4], 夏 鑫[3]

[1](西北工业大学 软件学院, 陕西 西安 710072)

[2](西北工业大学 自动化学院, 陕西 西安 710072)

[3](蒙纳士大学 信息技术学院, 澳大利亚 墨尔本)

[4](南通大学 信息科学技术学院 江苏 南通 226019)

[5](西北工业大学 空天地海一体化大数据应用技术国家工程实验室, 陕西 西安 710072)

[6](西北工业大学 大数据存储与管理工信部重点实验室, 陕西 西安 710072)

通讯作者: 吴潇雪, E-mail: wuxiaoxue00@gmail.com

摘 要:软件安全问题的发生在大多数情况下会造成非常严重的后果，及早发现安全问题是预防安全事故的关键手段之一。安全缺陷报告预测可以辅助开发人员及早发现被测软件中潜藏的安全缺陷从而尽早得以修复。然而，由于安全缺陷在实际项目中的数量较少，而且特征复杂（即安全缺陷类型繁多，不同类型安全缺陷特征差异性较大），这使得手工提取特征相对困难，并随后造成传统机器学习分类算法在安全缺陷报告预测性能方面存在一定的瓶颈。针对该问题，论文提出基于深度学习的安全缺陷报告预测方法，采用深度文本挖掘模型 TextCNN 和 TextRNN 构建安全缺陷报告预测模型，针对安全缺陷报告文本特征，使用 skip-grams 方式构建词嵌入矩阵，并借助注意力机制对 TextRNN 模型进行优化。所构建的模型在 5 个不同规模的安全缺陷报告数据集上展开了大规模实证研究，实证结果表明，深度学习模型在 80%的实验案例中都要优于传统机器学习分类算法，性能指标 F1-score 平均可提升 0.258，在最好的情况下甚至可以提升 0.535。除此之外，针对安全缺陷报告数据集存在的类不均衡问题，对不同采样方法进行了实证研究并对结果进行了分析。

关键词: 安全缺陷，安全缺陷报告预测，深度学习，文本挖掘

中图法分类号: TP311

中文引用格式: 郑炜,陈军正,吴潇雪,陈翔,夏鑫. 基于深度学习的安全缺陷报告预测方法实证研究.软件学报. http://www.jos.org.cn/1000-9825/ 0000.htm

英文引用格式: Zheng W, Chen J, Wu X, Chen X, Xia X. Empirical Studies on Deep Learning-Based Security Bug Report Prediction Methods. Ruan Jian Xue Bao/Journal of Software, 2016 (in Chinese). http://www.jos.org.cn/1000-9825/0000.htm

Empirical Studies on Deep Learning-Based Security Bug Report Prediction Methods

ZHENG Wei[1,5,6], CHEN Jun-zheng[1], WU Xiao-xue[2], CHEN Xiang[1,4], XIA Xin[3]

[1](School of Software, Northwestern Polytechnical University, Xi'an 710072, China)

[2](School of Automation, Northwestern Polytechnical University, Xi'an 710072, China)

[3](Information Technology, Monash University, Melbourne, VIC, Australia)

[4](School of Information Science and Technology, Nantong University, Nantong 226019, China)

[5](National Engineering Laboratory for Integrated Aero-Space-Ground-Ocean Big Data Application Technology, Xi'an 710072, China)

[6](MIIT Key Laboratory of Big Data Storage and Management, Northwestern Polytechnical University, Xi'an 710072, China)

*收稿时间: 0000-00-00; 修改时间: 0000-00-00; 采用时间: 0000-00-00; jos 在线出版时间: 0000-00-00
CNKI 在线出版时间: 0000-00-00
基金项目：陕西省科学技术研究发展计划(2015GY073)，陕西省科学技术研究发展计划项目(407085951067)，陕西省重点研发计划（2019GY-057）

Abstract: The occurrence of software security issues can cause serious consequences in most cases. Early detection of security issues is one of the key measures to prevent security incidents. Security bug report prediction (SBR) can help developers identify hidden security issues in the bug tracking system and fix them as early as possible. However, since the number of security bug reports in real software projects is small, and the features are complex (i.e., there are many types of security vulnerabilities with different types of features), this makes the manual extraction of security features relatively difficult and lead to low accuarcy of security bug report prediction with traditional machine learning classification algorithms. To solve this problem, we propose a deep learning-based security bug report prediction method. The text mining models TextCNN and TextRNN via deep learning are used to construct security bug report prediction models. For extracting textual features of security bug reports, the skip-grams method is used to construct a word embedding matrix. The constructed model has been empirically evaluated on five classical security bug report datasets with different scales. The results show that the deep learning model is superior to the traditional machine learning classification algorithm in 80% of the experimental cases, and the performance of our constructed models can improve 0.258 on average and 0.535 at most in terms of F1-score performance measure. Furthermore, we apply different re-sampling strategies to deal with class imbalance problem in gathered SBR prediction datasets, and the experiment results are discussed.

Key words: security issue, security bug report prediction; deep learning; text mining

近些年来，互联网技术的快速普及在推动各行各业快速发展的同时，也助长了各种网络安全事件的发生[1]。国际权威漏洞发布组织的 CVE（Common Vulnerabilities and Exposures）[2,3]数据显示，近年来系统安全漏洞数量急速上升：2016 年发现的安全漏洞为 6447 个，2017 年发现的安全漏洞为 14714 个，2018 年发现的安全漏洞则达到 16555 个。软件安全漏洞也称为安全缺陷，是软件缺陷的一种类型，这类缺陷的存在可能会导致系统被恶意攻击者利用，从而造成信息泄露、篡改、系统瘫痪等安全问题。缺陷报告记录软件开发、运行、以及维护过程中发现的各种问题，是缺陷调查与再现的主要依据。随着软件漏洞数量的增加以及安全漏洞类别的衍变，手动对缺陷报告进行分类变得越来越困难[4]。除了需要一定的人力，手动缺陷报告分类还要求分类人员具有较为丰富的安全领域知识。已有研究指出[4,5,8]，实际项目中，许多安全缺陷报告往往由于开发人员或者测试人员安全知识缺乏等因素未被及时标记为安全相关缺陷报告，从而长期滞留在缺陷报告库中，直到安全事件发生并造成严重的后果。针对此问题，研究人员尝试采用机器学习等方法进行安全缺陷报告自动预测[4-8]。

安全缺陷报告预测是指通过机器学习等方法从大量的缺陷报告库中识别出与安全相关的缺陷报告。给定一个缺陷报告（bug report， 简称 BR），如果该 BR 跟安全相关，则称其为安全缺陷报告 (security bug report，简称 SBR)；如果 BR 与安全无关，则称为之为非安全缺陷报告（non-security bug report，简称 NSBR）。从大量的 BR 中识别出 SBR 这一问题可以被形式化建模为二分类问题。基于该思想，研究人员开展了一系列研究工作。例如，Gegick 等人[8]将文本挖掘与分类算法相结合，采用朴素贝叶斯(Naïve Bayes)，k 最近邻（k-nearest neighbor）等分类算法对来自思科的实际工程项目的缺陷报告进行预测。Peter 等人[5]针对安全缺陷预测中样本不均衡问题，给出了一种噪音数据过滤框架 FARSEC，该框架可以用于过滤掉 NSBR 中与安全关键词相似度较高的样本（即缺陷报告）。在此基础上，Shu 等人[6]进一步采用 SMOTE 进行样本不均衡问题处理，并采用超参优化方法 SMOTUNED 进行参数优化。这些研究主要采用分类算法与文本挖掘技术相结合的方法，通过提取 BRs 中与安全相关的关键字和词频等特征，然后用于训练分类模型并对大量未标记样本进行预测。这些研究成果为 SBR 识别奠定了基础。但是，已有研究提出的方法在预测性能上还存在较大的提升空间。例如，Peter 等人[5]所给出的实验结果中，在论文主要采用的性能度量指标 G-measure 最优得分 53.8%~71.9%的情况下，其查全率仅为 47.6%~66.7%，而误报率却高达 3.0%～41.8%。

传统机器学习分类算法对特征工程较为敏感，由于安全缺陷的复杂性和稀有性，缺陷报告中安全相关特征提取困难，从而导致模型预测性能难以提升。基于神经网络的深度学习技术在许多任务中被证明优于传统的机器学习方法，已在计算机视觉，自然语言处理，语音识别等领域内的大量任务中取得较大研究进展[9,49,50,53,55]。据我们所知，论文首次将深度学习引入到 SBR 预测问题中，基于 BR 的文本描述信息来进行

SBR 预测，并随后使用对深度学习在 SBR 预测的效果进行实证研究。通过经典文本分类模型 TextCNN（Convolutional Neural Networks for Sentence Classification）[12]和 TextRNN（Recurrent Neural Network for Text Classification）[13]构建 SBR 预测模型，并采用注意力机制（Attention Mechanism）[52] 对 TextRNN 模型进行优化。同时，就深度学习模型对 SBR 预测的性能以及不同的采样策略对预测结果的影响进行了实证研究。

我们将基于 TextCNN、TextRNN 以及注意力机制构建的深度学习模型应用于 4 个小规模数据集（每个数据集包含 1000 条 BRs）和一个大规模数据集（包含 41298 条 BRs），并与传统分类算法进行比较。结果表明：论文构造的基于 TextCNN 和 Attention+TextRNN 的深度学习模型对 SBR 预测能力在 80%的情况下，其性能要显著优于四种传统机器学习模型朴素贝叶斯，逻辑回归，k 最近邻， 随机森林，性能指标 F1-score 平均可提升 0.258，最多时甚至可以提升 0.535。

论文的主要贡献可总结如下：

（1）据我们所知，论文首次提出了基于深度学习的 SBR 预测方法。根据 SBR 文本特征，基于经典文本分类方法 TextCNN、以及 TextRNN 与注意力机制相结合构建 SBR 预测模型。

（2）基于实际开源项目验证了论文所提方法的有效性。实证研究共分析了来自 Ambari、Camel、Derby、Wicket 和 OpenStack 这 5 个项目累计 45298 个 BRs。最终结果表明论文所提方法的性能要显著优于传统的机器学习分类方法。

（3）论文深入分析了类不均衡问题、不同词嵌入方法对模型性能的影响，从而为如何有效使用论文所提方法提供了指导。

论文剩余内容的组织结构安排如下：第 1 节介绍安全缺陷预测的研究背景和相关研究工作。第 2 节介绍深度学习模型构建具体过程和方法细节。第 3 节介绍论文的实证研究，包括研究问题、评测对象、评测指标、实验流程以及关键参数设置。第 4 节对实证研究结果进行详细分析和总结。最后总结全文，并对下一步研究工作进行展望。

1 相关工作

随着软件系统安全问题的日益突出，SBR 预测也受到越来越多的关注[4-8]。软件项目一般使用缺陷跟踪系统（如 JIRA、LaunchPad、Bugzilla 等）对项目开发和运行中产生的缺陷进行记录和跟踪。在长期积累的大量 BRs 中，有些 BRs 已被明确标记为 SBR（比如与 CVE 数据相关的缺陷），这些缺陷报告对于识别新的 SBR 具有非常重要的参考价值。本文主要研究如何使用深度学习进行 SBR 文本特征挖掘，从而更高效更准确的完成 SBR 预测。本节将从安全缺陷报告预测和基于深度学习的文本分类这两个角度对已有工作进行分析。

1.1 安全缺陷报告预测

SBR 预测是及早发现系统安全缺陷的关键方法，已成为当前软件工程领域一个研究热点[4-8]。安全缺陷预测可以尽早地从缺陷库中的大量缺陷报告中发现隐藏的安全缺陷，因此开发团队可以更为合理的分配资源使得安全问题尽早得以解决，从而有效避免后期可能造成的重大危害。近些年来，研究人员针对该问题提出了多种解决方法。

Gegick 等人[8]首先提出将文本挖掘的方法应用于 SBR 描述信息分析，通过已标记的 SBR 的 term-by-document 变量矩阵对 SAS 文本挖掘器进行训练，然后对未知样本（即未标记缺陷报告）进行预测。并对来自思科实际工业项目中收集的缺陷报告数据集进行了实际验证。随后，Wijayasekara 等人[4]对 Linux 内核和 MYSQL 这两个项目在 2006~2011 年期间暴露的安全漏洞进行了分析，指出 Linux 中 32%的 SBRs 和 MySQL 中 62%的 SBRs 都属于隐藏 SBRs，在缺陷数据库中未能被及时识别出，而且这一数字在接下来的两年仍有上升趋势。他们同时给出了基于文本分类的隐藏 SBRs 的识别方法。

最近 Peter 等人[5]提出了一种噪音数据过滤方法 FARSEC，首先使用 TF-IDF（Term Frequence–Inverse Document Frequence）从 SBR 提取安全相关关键词，然后计算 NSBR 跟安全关键词表之间的相似度得分，通过得分排序，将 NSBR 中与安全关键词表相似度较高的样本晒除。该方法一方面可以移除掉可能标记错误的

样本，另一方面可以对 NSBR 进行欠采样处理，从而有效减小正负样本间的数量差距。FARSEC 在 4 个小规模数据集（Ambari、Camel、Derby 和 Wicket）和一个大规模数据集（Chromium）上进行了实验验证，结果表明 FARSEC 可以在一定程度上提高 SBR 预测性能。随后，Shu 等人[6]通过 SMOTUNED[54]超参优化方法对其进行改进，并对分类算法参数优化和数据预处理方法参数优化对 SBR 预测性能的影响进行实验对比，结果表明数据预处理方法（即类不均衡处理方法 SMOTE）的参数优化要比分类算法参数优化对预测结果的影响更大，更有助于提高 SBR 预测的查全率。

与已有 SBR 预测研究工作不同，论文首次从深度学习入手，基于面向自然语言处理的经典文本分类模型 TextCNN 和 TextRNN 构建 SBR 预测模型。

1.2 基于深度学习的文本分类

文本分类属于自然语言处理的一个分支，随着深度学习在自然语言处理中的长足发展和不断应用，基于深度学习的文本分类技术也取得了显著进展[14]。基于深度学习的文本处理模型包括 TextCNN、TextRNN、FastText、TextRCNN、HAN 等。基于深度学习的文本分类中，一个非常重要的问题就是确定分类的网络。某些情况下，分类的文本包含多种语言，如果针对每一种语言都训练一个网络，会让问题变得更为复杂并降低了网络的可迁移性。Yang 等人提出了一种基于分词注意力模型 HAN（Hierarchical Attention Networks for Document Classification），可以有效解决上述问题。当样本数据量较小时或者正负样本数差距较小时，传统的卷积神经网络模型会因为得不到足够的样本训练或者池化层丢失了大量信息导致对输入变化不敏感，胶囊网络有效的解决了这些问题[21,22]。胶囊网络通过基于挤压函数的卷积来感受和传递特征，并用胶囊（向量）来描述，使得比传统卷积神经网络获取和保留了更多的信息。

词嵌入算法是深度学习在文本处理中的一个核心，良好的词嵌入算法可以很好的反映词语的相似关系，而这些关系又很难被传统的文本分类网络学习到，所以良好的词嵌入算法可以给模型性能带来大幅度提升。在确定词向量的维度时，大部分研究工作偏向于经验主义，因此缺乏理论依据。Yin 等人[23]提出了 PIP（Pairwise Inner Product）损失来描述词嵌入矩阵的维度理想度，并通过实验进行了验证。较早的词嵌入算法是 Mikolov 等人[24]提出的 word2vec，他们提出两个新的模型：连续词袋模型（Continuous Bag of Words，简称 CBOW）和连续 skip-gram 模型来训练词向量。实验结果表明：相对于基于潜在语义（Latent Semantic Analysis，简称 LSA）具有更好的表现；而相对于潜在狄利克雷分配（Latent Dirichlet allocation，简称 LDA）[25]算法，word2vec 方法可以大幅度减少工作量。但是在后续的使用中，研究人员发现 word2vec 依赖于大量的文本数据，在较小的数据集上一般表现不佳。因此 Bojanowski 等人提出了基于 word2vec 方法的字符级 n-grams[26]，将每个单词表示为字符的 n-gram。每个字符和一个向量相关联，而单词则表现为这些向量的和。该方法的优点是对于训练中未出现的单词也能计算出比较好的表示。他们在 9 种不同语言的数据集上测试，证明了这种方法达到了当时最好的性能。Pennington 等人[27]则提出了 Glove 方法，他们关注于词和词之间共现的统计数据，核心思想是认为两个词如果共同出现的频率越高，则这两个词的相关性越高。即如果词 a 和词 b 的关联度大于词 a 和词 c，则应有$P_{ab} > P_{ac}$，其中 P 代表一定距离内（窗口词）两个词共同出现的概率。但是有时候，同一个词在不同的语句中有着不同的意义（即多义词），因此 Peters 等人提出了 ELMo（Embeddings from Language Models）[28]，给出了预训练一个双向的语言模型，再根据语言模型的每一层计算出每个单词的词向量。

ELMo 的成功，让研究人员意识到大家发现了在非任务大语料文本中预训练的重要性。随后研究人员设计出两个重要网络，一个是 BERT[17]，另外一个是 GPT[29]。GPT 使用 Transformer[30]代替传统的 LSTM 网络，并且使用了非监督预训练和监督微调相结合的方式，在多项任务中（包括文本分类任务）达到了当时最高水平。并且该论文通过对比证明了预训练对分类任务有着显著的提升。GPT-2 作为一个通用的语言模型[29]，在高达 800 万个网页上训练而成，在很多任务上达到了专用模型的水平，并且部分任务的表现已经达到了人类的水平，可见在大量文本上预训练可能是未来的趋势。Google 提出的 BERT 是一个基于微调的表示模型，同样也使用了 transformer，并且使用上下文双向预测[31]来预训练一个语言模型（masked language model，简

称 MLM），其刷新了 11 项 NLP 问题的记录。此外，数据类别不均衡是影响分类准确率的因素之一[32,33]。Huang 等人对经典深度学习分类模型在不均衡数据集上的表现进行了实验验证，表明通过实施深层网络来维持集群间和边际，可以学习更具辨别能力的深层模型。Song 等人[34]提出基于聚类的双向采样方法来进行数据类别不均衡问题处理。

论文首次将深度学习引入 SBR 预测问题，根据 SBR 文本描述信息差异性大、安全特征稀缺的特征、SBR 预测数据集中正样本（SBR）较少导致类别严重不均衡（例如在 Peter 等人[5]搜集的 5 个数据集中，其正样本所占比例为 0.5%～9.0%）等问题，采用 word2vec 的 skip-gram 模型（论文将窗口大小设置为 3）来训练词向量，通过设置全连接层偏置项（bias）提高模型泛化能力；同时，结合实际项目经验进行句长设置，构建出符合 SBR 文本特征的分类模型。

2 基于深度学习的安全缺陷报告预测方法

2.1 方法的整体框架

基于深度学习的 SBR 预测方法框架如图 1 所示，其共包含四个阶段：

（1）阶段 1 执行 BR 数据集的收集与划分，具体细节见 2.2 节。

（2）阶段 2 执行文本预处理。该阶段将文本数据预处理成数字序列，深度学习模型可以方便的根据序列获得到单词的表示(词向量)，具体细节见 2.3 节。

（3）阶段 3 进行深度学习模型的构建。我们一共选择了两种适合文本挖掘的深度学习模型，分别是 TextCNN 和 TextRNN [46]。这两个模型分别衍生自经典的 CNN 模型和 RNN 模型，选用这两个具有代表性的模型可以确保我们实证研究的结论更有代表性。具体细节见 2.4 节。

（4）阶段 4 借助已经构建的深度学习模型，从未标记的缺陷报告中预测中 SBR。

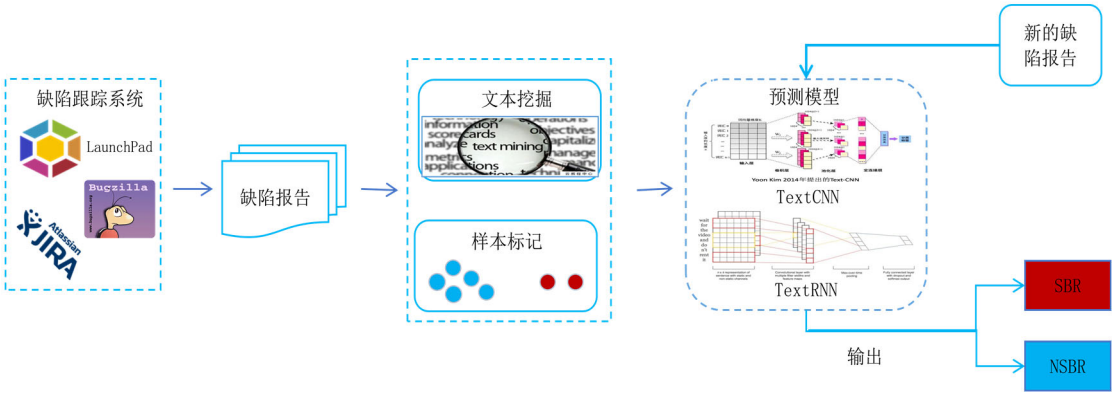


Fig.1 Framework of deep learning-based security bug report prediction

图 1 基于深度学习的安全缺陷报告预测方法框架

2.2 阶段I：缺陷报告的收集

项目开发和运维过程中产生的缺陷，都会使用特定的缺陷跟踪系统（例如 JIRA、Bugzilla、LaunchPad 等）进行管理。BRs 从多个维度对所发现的问题进行记录，表 1_是 LaunchPad 系统中 OpenStack 项目的一个 BR 示例，该示例展示了 BR 中的关键字段信息。在 BR 的多个字段中，缺陷描述信息（Bug Description）是

_在 LaunchPad 中，OpenStack 项目使用 Tag 标识特殊的 BR 类型，如 security、performance 等。同时由于 Description 信息较长，本示例只显示了前面一部分文字信息

对缺陷最详尽的描述，一般包括缺陷触发的条件、问题现象、预期结果等文本描述信息。如其它 SBR 预测的研究[5,6]一样，本论文也使用 BR 的描述信息进行 SBR 预测。虽然开源缺陷跟踪系统收集了大量的 BRs，可以提供丰富的数据资源。但是，提交缺陷报告的人员质量和知识背景差异较大，因此在收集数据集过程中，我们需要对一些低质量缺陷报告（比如没有缺陷描述信息的）进行过滤。 Chaparro 等人[36]开发出一种过滤信息不完整缺陷报告的工具，该工具认为缺陷报告描述信息中，OB（observed behavior）、S2R（steps to reproduce）和 EB（expected behavior）是缺陷报告描述的基本内容，如果这三者缺少任何一个，则认为该缺陷报告的描述信息不完整，因此可以通过自然语言处理和解析方式自动进行判断和过滤。本论文考虑的数据集，一部分来源于已有的广泛使用的安全缺陷报告数据集；另一部分是在作者前期工作[35]的基础上进一步完善得到的 OpenStack 安全缺陷预测数据集。在该数据集的收集过程中，我们同样采用了 Chaparro 等人[36]的方法对描述信息不完整的缺陷报告进行了过滤。

Table 1 A bug report example from OpenStack project

表 1 来自 OpenStack 项目中的一个缺陷报告示例

字段	内容
Key	1218977
Title	DOS by passing an ephemeral or swap of arbitrary size
Status	Fix Released
Importance	Critical
Tag	Security
Description	Due to a previous bug that was never caught and the fact that we can now pass ephemeral and block devices through the API, it is possible to ask nova to create an arbitrarily large ephemeral block device - which nova will happily do (and by default make it raw). The bug was introduced in commit 0ef7e15e225efcce3e02098cb1d57f9f40181f82 as before that commit the ephemeral device size will be defaulted to whatever was in the instance_type - due to a bug this defaulting was not done anymore (see compute.api.API._update_block_device_mapping). Steps to reproduce: ndipanov@localhost devstack]$ nova flavor-show 1 …

2.3 文本预处理

论文在文本预处理时依次执行如下四个步骤。

（1）步骤 1 移除停用词。停顿词是指对问题分析没有实际价值的一些词语，如‘this’、‘that’、‘we’、‘then’、‘a’等。移除停顿词一方面可以降噪，另一方面可以减小数据量，降低计算开销和空间开销。我们首先使用 NLTK[19]提供的通用的停用词集。但是 NLTK 提供的默认停顿词对于缺陷报告问题的分析并不完善，我们通过人工分析，在此基础上进一步对论文所使用的停顿词进行了补充。

（2）步骤 2 进行句子填充（Sentences padding）。 深度学习模型在进行文本数据处理的时候，需要执行句子填充操作（即统一句子长度）。一般文本处理采用最长句子的长度，但是，由于 BR 的描述信息长度差异较大，如果全部依据最长句子进行填充，一方面会大幅度增加模型的空间开销和时间开销，另一方面也会引入大量噪音。因此，我们根据实际预测效果，选择平均长度的 1.5 倍作为最大句子长度，其计算公式为：

$$L = ceil(L_{avg} \times 1.5) \tag{1}$$

其中，L_{avg}代表句子平均长度，$ceil$ 代表向上取整。长度大于该计算结果的句子会被按序截取，即第 L 个词以后的都会被截断。长度不够的句子会自动补充无意义符号（如 0），使其长度达到 L。

（3）步骤 3 是建立词汇-索引映射表。词表是一个包含所有出现过的词和一个特殊符号的集合，每种出现过的词和词表的一个元素有着唯一的对应，并且所有未出现的词都将会和词表中的特殊符号对应。

（4）步骤 4 是建立词嵌入(Word Embedding)矩阵。词嵌入矩阵是一个$S \times d$ 大小的矩阵，其中 S 是词表中的元素个数，d 是每个词的词嵌入向量的维度。矩阵的每一个行向量都代表了一个词，并且按顺序和词表

中的每一个词相对应。Word2vector 提供 skip-grams 和 CBOW 两种词嵌入矩阵构建方法，本论文中，我们使用 skip-grams 方法，并根据安全缺陷文本特征设定窗口大小为 3，完成词嵌入矩阵的构建。

这样，我们数据集中的每一条记录的描述信息都成为了长度为 L 的向量。所有的缺陷描述构成了一个$T = L \times N$的矩阵，其中 L 是句子长度，N 是样本数。样本的标签构成了一个长度为 N 的列向量 La。设存在一个理想模型f，则有$La = f(T)$。

2.4 深度学习模型

论文主要考虑了两种深度学习模型 TextCNN 和 TextRNN，并采用注意力机制对模型进行优化。

2.4.1 TextCNN

TextCNN 是一个由经典的卷积神经网络（CNN）衍生出来的用于文本分类的神经网络，其结构如图 2 所示。

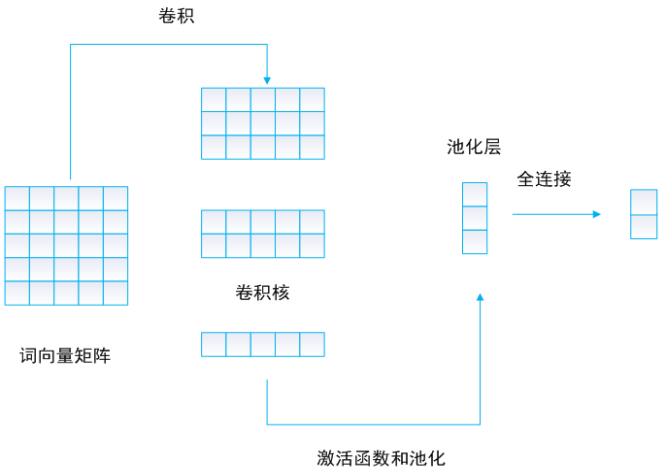


Fig. 2 Network Structure of TextCNN

图 2 TextCNN 网络结构

模型先在词向量矩阵上用不同大小的卷积核卷积，每个卷积核类似于一个可训练的滤波器，提取输入的部分特征，然后将卷积结果经过激活函数（这里我们用的是 ReLU）处理后输入到池化层，池化层会生成更高维的向量。然后将池化层的结果通过 dropout 层随机选择后（这样可以一定程度避免过拟合）通过全连接层连接，最后得到分类结果。

CNN 的空间问题复杂度如下所示，其与输入数据本身的大小无关，主要涉及模型的参数数量和每层输出的特征图大小：

$$\text{Space} \sim O\left(\sum_{l=1}^{D} K_l^2 \times C_{l-1} \times C_l\right) \tag{2}$$

其中，K 为卷积核的尺寸，C 为通道数，D 为网络的深度。

针对安全缺陷报告本身特征，我们对模型做了一些针对性的设计。首先，我们设置了多个卷积核，大小分别是$i \times d, i = 1,2,3$，其中 d 是每个词的词嵌入向量的维度。这种处理方式可以把卷积视野抽象成i（$i = 1,2,3$）个单词。其核心公式为$y = f(C_1 + C_2 + C_3)$，其中，C_1，C_2，C_3 是不同大小卷积核的输出结果。

2.4.2 TextRNN

TextRNN 是本论文选择的另一种深度学习网络模型，也是经典的用于文本处理的深度学习模型，基于 RNN 的基本思想是利用给定序列中存在的信息（如文本）。给定一系列单词，将单词进行数字化表示（GloVe 或 Word2Vec）馈送到神经网络并计算输出。在计算下一个单词的输出时，还会考虑前一个单词的输出。RNN

之所以被称为循环，是因为它们使用先前计算的输出对序列的每个元素执行相同的计算。在任何步骤，RNN 执行以下计算，

$$RNN(t_i) = f(W \times x_{t_1} + U \times RNN(t_{i-1})) \tag{3}$$

其中，W 和 U 是模型参数，f 是非线性函数。$RNN(t_i)$是第 i 个 timestep 的输出，可以按原样使用，也可以再次输入参数化结构，如 softmax[37]，具体取决于当前执行的任务。训练是通过根据全部或部分 timestep 的输出制定损失目标函数，并尽量减少损失来完成的。我们使用基于 BiLSTM（Bi-Long Short-Term Memory）的 TextRNN 网络，通过在经典的 RNN 上加入三扇门（即输入门、输出门和遗忘门）而实现，其结构如图 3 所示：

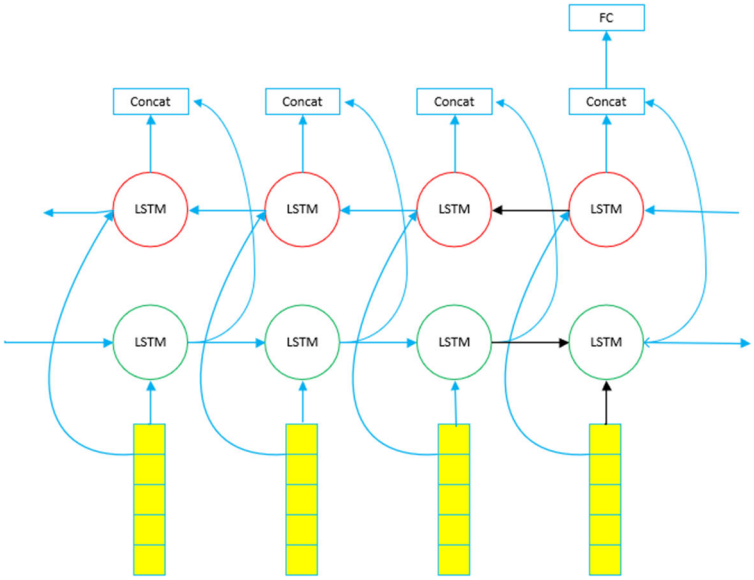


Fig. 3 Structure of TextRNN

图 3 TextRNN 的结构图

输入门、输出门和遗忘门可以让信息选择性的通过，各层主要通过一个 sigmoid 的神经层和一个逐点相乘的操作实现。其中，sigmoid 层的输出是一个向量，向量的每个元素取值介于[0,1]之间，表示让对应的信息通过的权重（例如：0 表示不允许通过，1 表示允许所有信息通过）。

输入门:决定下一步允许多少新的信息加入到 cell 状态中，由“input gate layer”的 sigmoid 层决定哪些信息要更新，其公式为：

$$i_t = \sigma\ (W_i * [h_{t-1}, x_t] + b_i) \tag{4}$$

其中，σ是激活函数，h_{t-1}是前一时刻隐层状态，x_t是当前时刻的输入，W_i和b_i是线性关系的系数。

输出门: 确定最终输出值。输出状态是一个过滤后的状态。运行一个 sigmoid 层来确定细胞状态的哪个部分将会输出，然后，把细胞状态通过 tanh 进行处理，得到一个[-1,1]之间的值，并将它和 sigmoid 门的输出相乘，最终仅仅会输出确定要输出的部分：

$$h_t = \sigma\ (W_o[h_{t-1}, x_t] + b_o) * tanh(C_t) \tag{5}$$

其中，σ是激活函数，h_{t-1}是前一时刻隐层状态，x_t是当前时刻的输入，W_o和b_o是线性关系的系数。

遗忘门：LSTM 第一步决定从细胞中遗忘的信息内容，通过遗忘门来实现。该门会读取h_{t-1} 和x_t，输出一个[0, 1]之间的数值给每个在细胞状态：

$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] + b_f\right) \tag{6}$$

其中，W_f和b_f是线性关系的系数，σ是激活函数，h_{t-1}是前一时刻隐层状态，x_t是当前时刻的输入。LSTM 有多个变种[37]，有着不同的适应场合和性能。图 3 中可以看到，我们将词向量输入到双向的 LSTM 中，然后将结果进行连接，最后通过全连接层（Full Connection，简称 FC）得到分类结果。我们使用双向 LSTM 的原因是双向 LSTM 可以有效缓解越往后的单词对结果影响越大的问题。

2.4.3 注意力机制

论文进一步借助注意力机制（Attention Mechanism）对 TextRNN 模型进行优化。SBR 的识别主要依赖 BR 中的描述信息（Description），采用自然语言描述形式，其安全相关文本特征较为稀疏，给 SBR 预测准确性带来一定困难。注意力机制源于对人类视觉的研究，使训练重点集中在输入数据的相关部分，忽略无关部分[52,53]。注意力机制最早由 Bahdanau 等人[52]提出并应用于机器翻译任务中。其基本思想是基于 encode-decode 模式，首先定义 RNN 模型中的条件概率，其计算方法如公式（7）所示：

$$p(y_i \,|y_1, \ldots, y_{i-1}, x) = g(y_{i-1}, s_i, c_i) \tag{7}$$

其中，s_i为 RNN 在i 时刻的隐藏状态，其计算方法如公式（8）所示：

$$s_i = f(s_{i-1}, y_{i-1}, c_i) \tag{8}$$

c_i为上下文向量，其计算方法如公式（9）所示：

$$c_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j \tag{9}$$

α_{ij}为每个注解h_j 的权重，计算方法如公式（10）：

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{T_x} \exp(e_{ik})} \tag{10}$$

其中$e_{ij} = a(s_{i-1}, h_j)$ 是所得到的对齐矩阵，用于记录位置 j 的输入和位置 i 的输出的匹配程度。简单而言，注意力机制把源语言端的每个词学到的表达和当前要预测的词联系了起来，在模型训练好后，根据注意力矩阵得到源语言和目标语言的对齐矩阵。

3 实验设计

为了支持我们的结论，我们设计了如下四个研究问题（Research Question，简称 RQ）来指导我们的实验设计。

RQ1：深度学习方法与已有基准方法相比是否可以提升安全缺陷报告预测的性能?

RQ2：针对安全缺陷报告预测问题，不同的正负样本比例对深度学习模型性能的影响如何？

RQ3：不同的深度学习模型的迭代表现是否相似？

RQ4：词嵌入方法对基于深度学习的安全缺陷预测方法的性能影响程度如何？

随后该节对实证研究中所需要的数据集、性能评价指标进行介绍，最后给出实验流程及关键参数的具体设定。

3.1 评测对象

论文在实证研究中使用了两部分数据集。第一部分是 Ohria 等人[38]提供的 4 个小规模数据集，分别来自四个不同的项目：Ambari[39]、Camel[40]，derby[41]和 Wicket[42]。这 4 个项目都来自于 Apache，并使用 JIRA 作为缺陷跟踪系统。Ambari 给 Apache Hadoop[43]集群提供管理服务和可视化的用户接口。Camel 简化了特定领域语言的路由和中介规则的建立。Derby 是一个完全由 Java 编写的轻量级的数据库。Wicket 是一个基于 Java 的 web 开发框架，这 4 个项目代表四类典型应用系统。Ohria 等人[38]从 4 个项目中各选择了 1000 个具有较大影响的缺陷，并采用人工检查和确认的方式进行了标注，标注类型包括 6 类：surprise，dormant，blocking，security，performance 和 breakage。Peter 等人[5]在此基础上根据 security 标签提取了 4 个用于安全缺陷预测的数据集，并将其公开。因此，本论文直接使用了他们处理之后的 4 个小规模数据集。

本论文使用的另外一个数据集是我们自己基于 OpenStack 项目缺陷报告构建的一个大规模数据集，包含 41298 多个来自 OpenStack 项目的缺陷报告。Openstack 是一个由 NASA 和 Rackspace 合作发起的开源云管理平台[44]，包含多个用于构建云平台的组件设施（如 Nova、Neutron、Swift、Cinder 等）。我们在之前的研究工作中[35]对 OpenStack 的 41298 个缺陷报告从不同维度进行了深入分析，包括缺陷类型分布、缺陷、解决措施等。本论文是在原有工作基础上，进一步结合 OpenStack 的 CVE 数据进行了安全样本标记，从而构建了用于安全缺陷报告预测的 OpenStack 数据集。OpenStack 的缺陷报告使用 LaunchPad[45]进行跟踪管理，至今已积累了超过 14w 条缺陷报告。在之前的研究工作[35]中，我们确定了如下的缺陷报告筛选条件：

- Importance 字段仅考虑 Critical、High 和 Medium。
- Status 字段仅考虑 Fix Committed 和 Fix Released。
- Report data 字段的时间要早于 2018 年 5 月 1 日。

我们采用这些筛选条件，一方面因为符合这些条件的数据在实际项目中开发人员的关注度更高；另一方面这些缺陷报告都是已修复完成的并且经过了分析过程，其结论可信度较高。根据这三个条件，总共检索到 48255 条缺陷报告，在此基础上，我们采用 Chaparro 等人[36]提供的工具对描述质量较低的缺陷报告进行了进一步过滤，最终得到 41298 条符合条件的缺陷报告。最后在此基础上，根据以下两条规则进行安全相关缺陷报告的标记：

（1）规则 1：BR 与 CVE 记录相关[3]。CVE 是国际权威漏洞跟踪与发布组织，OpenStack 系统目前在 CVE 系统中约有 180 条记录。为了便于了解漏洞详细记录，对于开源系统，CVE detail[3]网站会将每个漏洞所对应的原始缺陷报告进行关联。因此，当 OpenStack 在 LaunchPad 中的缺陷报告与 CVE 数据关联，则表明该缺陷是安全相关缺陷。

（2）规则 2：缺陷报告在 LaunchPad 系统中本身被标记了 security 的 label。在 LaunchPad 系统中，OpenStack 的 SBR 通过将‘label’字段设置为‘security’来进行标记的，因此，论文将该部分 BRs 也作为 SBR 的一部分。

当满足这两条规则的任意一条时，我们认为这个缺陷报告是安全相关缺陷报告。最终，本论文实证研究采用的数据集的统计信息如表 2 所示，主要包括数据集的名称、安全相关的缺陷报告数和安全无关的缺陷报告数。

Table 2 Statistics of security bug report prediction datasets

表 2 安全缺陷报告预测数据集的统计信息

数据集名称	安全相关的缺陷报告数	安全无关的缺陷报告数
Ambari	29	971
Camel	32	968
Derby	88	912
Wicket	10	990
Openstack	239	41059

3.2 评测指标

给定测试集中的任意一个 BR，其可能的结果有 4 种：TP（True Positive）、TN（True Negative）、FP （False Positive）、FN（False Negative），反映了预测结果和实际结果的关系，可用混淆矩阵来表示，具体如表 3 所示。

Table 3 The confusion matrix of security bug report prediction

表 3 针对安全缺陷报告预测问题的混淆矩阵

预测结果 / 实际结果	SBR	NSBR

SBR	TP	FN
NSBR	FP	TN

基于上述混淆矩阵，我们依次介绍论文使用的评测指标。

准确率（Accuracy）：反映了模型预测的正确率，其计算公式为：

$$accuarcy = \frac{TP+TN}{TP+TN+FP+FN} \tag{11}$$

但是在类别不均衡的问题中使用 accuracy 可能无法客观反映模型性能。假如某个模型在大部分情况下都将少数类识别为多数类，但是因为少数类占比过少，所以即使准确率很高，这样的模型在实际任务中很难对少数类做出精确的预测。

查全率（recall）：预测为 SBR 的样本数量占实际 SBR 样本数量的比率，其计算公式如下：

$$recall = \frac{TP}{TP+FN} \tag{12}$$

查准率（precision）：预测正确的 SBR 数量占预测为 SBR 的数量的比率，其计算公式如下：

$$precision = \frac{TP}{TP+FP} \tag{13}$$

F1-score: 查全率和查准率可以说是一对相反的指标，查全率高说明模型更加倾向于将一个样本认为是正样本，而查准率高则说明模型在预测为 SBR 时更加谨慎。F1-score 则综合考虑查全率和查准率，其计算公式为:

$$F1-score = \frac{2*precision*recall}{precision+recall} \tag{14}$$

3.3 统计分析方法

论文使用 Wilcoxon rank-sum 检验[47]来分析深度学习方法与四种传统分类器之间在 SBR 预测上性能是否具有统计差异性。Wilcoxon rank-sum 是一个非参假设检验方法，用来检测两个数据集是否来自于相同分布的总体。对于相比较的两个数据集，零假设是两种方法之间不存在显著差异。如果由 Wilcoxon rank-sum 检验得到的 p-value 小于显著性水平，则拒绝原假设。也就是说，两种方法之间的差异被确定为具有统计显著性。论文使用 0.05 作为显著性水平，当 p-value <0.05 时，认为两组数据集具有统计显著性[49]。论文计算深度学方法与传统分类方法性能指标 F1-score 值的 Cliff's delta 来度量二者之间差异性。Cliff's delta 计算公式如下：

$$d = {2W}/{mn} - 1 \tag{15}$$

其中，W是 Wilcoxon rank-sum 检验统计值，m 和 n 分别为两组参与对比数据集的大小。 $W = R - n(n+1)/2$，其中 R 为 Cliff's delta 值的范围。Cliff's delta 的级别和取值范围如表 4 所示，总共分为 4 个级别。

Table 4 The level and range of Cliff's delta[47]

表 4 Cliff's delta 的级别与取值范围对应关系[47]

级别	Cliff's delta 的取值范围
Negligible	$\|d\| < 0.147$
Small	$0.147 \leqslant \|d\| < 0.33$
Medium	$0.33 \leqslant \|d\| < 0.474$
Large	$0.474 \leqslant \|d\|$

3.4 基准方法

为了验证深度学习对于安全缺陷报告预测的性能表现，我们将本文构建的深度学习预测效果与 Peter 等

人[5]近期发表于 TSE 的方法 FARSEC 进行对比。该方法使用文本挖掘技术与传统分类算法相结合进行安全缺陷报告识别，并给出一个框架 FARSEC 用于噪音数据过滤。该方法首先使用特征提取方法 tf-idf 从安全缺陷报告中识别与安全相关的关键词，之后，通过计算缺陷报告与安全关键词之间的相似度得分来对缺陷报告进行排名，取出排名最高的缺陷报告为噪音数据，将其从数据集中删除。在实验验证中，其所选定的关键词数量设置为 100 个，阈值 0.75 用于过滤噪声错误报告的错误报告得分。他们评估了 FARSEC 对 5 个数据集（即 Ambari、Camel、Derby、Wicket 和 Chromium）的有效性。该方法所使用到的分类算法包括：朴素贝叶斯，逻辑回归，k 最近邻，随机森林：

(1) 朴素贝叶斯（Naive Bayes，简称 NB）：朴素贝叶斯是基于贝叶斯法则的概率模型。其假设各个特征之间相互独立，所以逻辑简单，计算开销小。但是有时候变量之间并不是完全随机的，因此会损失一部分准确度。
(2) 逻辑回归（Logistics regression，简称 LR）：逻辑回归在线性回归模型的基础上经过激活函数（非线性函数，一个典型的例子就是 sigmod），使得回归的结果变成了分类结果。
(3) k-近邻（k-Nearest Neighbor，简称 KNN）。k-近邻算法基于一个简单的假设：如果一个样本的 k 个最相似的样本的大多数属于某一类别，则这个样本也属于那一类别。
(4) 随机森林（Random Forest，简称 RF）：随机森林是多棵分类树得到的。其中每棵分类树都由随机的样本和随机的特征训练而成。由于其引入了两个随机因素，所以不容易过拟合，并且对不均衡样本的处理效果较好。

3.5 实验流程和方法参数设置

数据选取和划分。实验主要涉及两组，第一组是在数据不做任何采样情况下，将深度学习模型与传统方法对 SBR 预测的性能进行对比。首先，在 5 个数据集上分别执行论文考虑的 2 种深度学习模型和 4 种基准方法，通过实验对比验证深度学习模型对 SBR 预测的有效性。第二组实验是验证不同样本比例对深度学习模型的影响，通过对训练数据集进行重新采样，探索 SBR 数量与 NSBR 数量呈不同比例情况下对深度学习预测性能的影响。首先，每个数据集中的样本按照对正负样本进行分层采样的方法随机分为 3 部分：训练集，验证集和测试集，分配比例根据工业项目模型使用经验进行设置，训练集样本数、验证集样本数及测试集样本数的比例为 6:2:2。 我们使用N_{sbr}表示 SBR 的数目，N_{nsbr} 表示 NSBR 数目，对于 4 个小规模数据集 Ambari、Camel、Derby 和 Wicket，我们使用“欠采样”方法，根据拟定正负样本比例设置随机选择相应数量的负样本。4 个小数据集在训练样本中的比例设置为： $N_{nsbr}:N_{sbr}=x$ ，$x=1,5,10$。对于大规模数据集 OpenStack，因为其正负样本数量差距较大（约 1：200），若按照欠采样方法进行处理，大量的负样本将会被丢弃，从而很可能丢失大量有用的特征文本信息，并影响预测结果。因此，针对 OpenStack 的样本分类不均衡问题，论文采用“过采样”方法进行处理，通过随机复制正样本信息来提高训练数据集中正样本分布比例。

模型参数的选择。论文基于开源机器学习框架 PyTorch[50]进行代码实现。对于 4 个小规模数据集，因为样本量比较少，综合考虑，每一批(batch) 30 个样本，并且训练 1000 次迭代（即 Epoch），而大数据集 OpenStack，每一批（batch）100 个样本，训练 1000 次迭代。然后，使用训练集对模型进行训练，并通过验证集评价模型，最终确定学习率为 0.01。由于 SBR 预测数据集中正样本（SBR）数量较少，存在严重类别不均衡问题（如本文所选择的 5 个数据集中，正样本所占比例在 0.58% ~ 9.00%之间）；此外，4 个小规模数据集由于正样本数量较少，导致所构建模型较小，因此，模型实现中我们对全连接层偏置项（$bias$）进行了调整以提高模型泛化能力。全连接层公式为$y=wx+bias$，其中 w 是权重，$bias$ 为偏置项，是全连接层的一种重要参数。我们测试了多组 $bias$ 值，发现当 $bias$=0.2 时，其构建的模型在 4 个小规模数据集上都有较好表现，因此，在本文实验中，我们将 $bias$ 的取值设置为 0.2。

模型训练和结果输出。对两个模型（TextCNN，Attention+TextRNN）在 5 个不同的数据集进行实验，每训练一个迭代（Epoch）就使用测试集进行测试，获得各个度量指标值。

4 结果分析

4.1 针对RQ1的结果分析

RQ1：深度学习方法与已有基准方法相比是否可以提升安全缺陷报告预测的性能?

目前,研究人员针对安全缺陷报告预测问题已经提出了基于各种传统分类算法与文本挖掘相结合的方法，本论文中我们以 Peter 等人[5]的工作作为基准方法，将本论文给出的深度学习模型的预测性能与其工作中的最优分类算法的性能进行比较,分别分析深度学习方法在小规模数据和大规模数据情况下对安全缺陷报告预测的性能表现。我们使用 F1-score 作为主要判断指标，同时也给出 Accuracy、Recall 和 Precision 的值。为了使实验结果更为公正、客观，我们对每一组实验运行 10 次，然后求平均值。表 5 是 5 个数据集不进行任何采样时传统分类算法和深度学习模型取得最佳 F1-score 指标的结果对比。第二列“传统分类算法”表示 4 个 baseline 分类器中表现最好的 F1-score，第三列“深度学习”取 TextCNN 和 Attention+TextRNN 在该数据集中得到的 F1-score 的较大值，最后一列给出了深度学习模型最优结果相对于 4 个传统分类模型对 F1-score 的提升值。

Table 5 Comparision of the best F1-score for traditional classifier and deep learning based method

表 5 传统分类算法和深度学习方法的最优 F1-score 对比

数据集名称	传统分类算法		深度学习方法		F1-score 提升
	算法名称	F1-score	算法名称	F1-score	
Ambari	NB	0.125	Attention+TextRNN	0.615	0.490
Camel	LR	0.108	TextCNN	0.643	0.535
Derby	NB	0.360	Attention+TextRNN	0.464	0.104
Wicket	KNN	0.048	TextCNN 或 Attention+TextRNN	0.000	-0.048
OpenStack	NB	0.201	Attention+TextRNN	0.410	0.209

在 5 个数据集中，有 4 个数据集中深度学习模型的性能优于传统分类算法。在 Wicket 数据集上，尽管深度学习传统分类算法差，但是实际上传统分类算法也只是正确识别出了一个正样本（TP=1，F1-score=0.048），因此两者性能差异较小。

为了验证深度学习方法的统计显著性，我们将表 3 中所给出的每个数据集上表现最佳的传统分类算法和深度学习算法分别运行 10 次，通过深度学习模型所得到的结果与传统分类算法所得结果进行对比，计算 Wilcoxon rank-sum 检验所得 p-value 和 Cliff’s delta，所得结果如表 6 所示。

Table 6 Comparison results in terms of p-value and Cliff’s delta

表 6 基于 p-value 和 Cliff’s delta 的比较结果

数据集名称	p-value	Effect Size
Ambari	<0.05	1.000(Large)
Camel	<0.001	1.000(Large)
Derby	<0.05	0.721(Small)
Wicket	<0.05	- 0.745(Large)
OpenStack	<0.05	1.000(Large)

通过表 5 和表 6，可以发现深度学习对 SBR 的预测性能在 80%的场景中优于传统分类算法，不论是在小规模数据集，还是大规模数据集 OpenStack，深度学习模型的预测性能都要显著优于传统分类算法。

4.2 针对RQ2的结果分析

RQ2：针对安全缺陷报告预测问题，不同的正负样本比例对深度学习模型性能的影响如何？

在文本分类问题中，不同类型样本的比例会影响最终预测结果准确率[31]。因此在这个 RQ 中，我们拟分析采用深度学习模型进行 SBR 预测问题中，不同正负样本比例对模型的预测性能可能产生的影响。我们对 4 个小规模数据集和 1 个大规模数据集采用不同的采样策略，并计算使用 TextCNN 和 Attention+TextRNN 预测模型时所得的 F1-score 值。对于 4 个小规模数据集 Ambari、Camel、Derby 和 Wicket，我们采用欠采样方法，随机减少训练样本中的负样本数量，使其样本比例达到拟定比例设置 $N_{nsbr}:N_{sbr}=x$ ，$x=1,5,10$，其结果如表 7 到表 10_所示。为了分析不同深度学习模型的迭代表现，我们打印了两个模型在整个实验过程中（共 1000 次迭代）F1-score 随着迭代进行的变化情况，表中所给出的是 1000 次迭代中所取得的最优 F1-score，同时，我们在表格第三列给出了取得最优值的迭代次数。

Table 7 Results on Ambari dataset

表 7 Ambari 数据集结果

算法名称	样本比例（N_{nsbr}：N_{sbr}）	最佳迭代次数	准确率（Accuracy）	查全率（Recall）	查准率（Presion）	F1-score
TextCNN	1	50	0.873	0.640	0.271	0.381
	5	444	0.957	0.400	0.833	0.541
	10	147	0.966	0.440	1.000	0.611
Attention + TextRNN	1	13	0.922	0.480	0.387	0.429
	5	52	0.959	0.440	0.786	0.564
	10	6	0.966	0.440	1.000	0.611

Table 8 Results on Camel dataset

表 8 Camel 数据集结果

算法名称	样本比例（N_{nsbr}：N_{sbr}）	最好结果出现的迭代次数	准确率（Accuracy）	查全率（Recall）	查准率（Presion）	F1-score
TextCNN	1	933	0.866	0.529	0.084	0.145
	5	794	0.976	0.412	0.438	0.424
	10	185	0.981	0.529	0.563	0.545
Attention + TextRNN	1	937	0.870	0.471	0.078	0.134
	5	12	0.981	0.412	0.583	0.483
	10	9	0.968	0.353	1.000	0.522

Table 9 Results on Derby dataset

表 9 Derby 数据集结果

算法名称	样本比例（N_{nsbr}：N_{sbr}）	最好结果出现的迭代次数	准确率（Accuracy）	查全率（Recall）	查准率（Presion）	F1-score
TextCNN	1	271	0.884	0.432	0.390	0.410
	5	31	0.927	0.405	0.682	0.508
	10	10	0.909	0.351	0.520	0.419
Attention + TextRNN	1	18	0.886	0.432	0.400	0.416
	5	12	0.929	0.459	0.680	0.548
	10	8	0.898	0.486	0.462	0.474

_ N_{nsbr} 表示参与实验的 NSBR 数量；N_{sbr} 表示参与实验的 SBR 的数量。

Table 10 Results on Wicket dataset

表 10 Wicket 数据集结果

算法名称	样本比例（N_{nsbr}：N_{sbr}）	最好结果出现的迭代次数	准确率（Accuracy）	查全率（Recall）	查准率（Presion）	F1-score
TextCNN	1	449	0.078	1.000	0.011	0.022
	5	958	0.781	0.625	0.029	0.055
	10	47	0.988	0.125	0.333	0.182
Attention + TextRNN	1	0	0.023	1.000	0.010	0.020
	5	0	0.023	1.000	0.010	0.021
	10	41	0.978	0.125	0.091	0.105

在大规模数据集 OpenStack 上，我们采用“过采样”方法进行正负样本不均衡问题处理，根据拟定比例对正样本进行随机复制。我们选择的复制比例分别为 1 倍、2 倍和 3 倍，其执行结果如表 11 所示。

Table 11 Results on OpenStack dataset

表 11 OpenStack 数据集结果

算法名称	正样本数量	最好结果出现的迭代次数	准确率（Accuracy）	查全率（Recall）	查准率（Presion）	F1-score
TextCNN	复制 1 倍	86	0.997	0.607	0.773	0.680
	复制 2 倍	330	0.997	0.571	0.842	0.681
	复制 3 倍	23	0.997	0.571	0.800	0.667
Attention + TextRNN	复制 1 倍	12	0.996	0.321	0.818	0.462
	复制 2 倍	36	0.994	0.250	0.467	0.326
	复制 3 倍	6	0.993	0.036	0.250	0.063

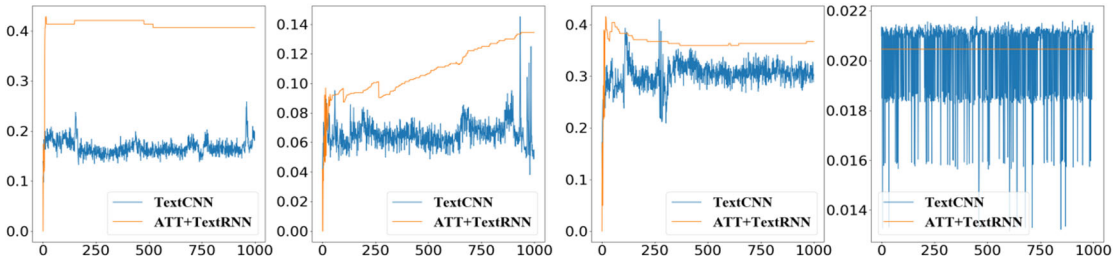
从表 7 到表 10，我们可以发现，在四个小规模数据集中，N_{nsbr}：N_{sbr} 比例越小（即不均衡程度越低），性能指标 Recall 的取值越大；但是，Precision 指标的取值却恰恰相反，并最终造成 F1-score 指标的取值也呈减小趋势。与原始数据集（未进行采样）执行结果（即表 5）相比，“欠采样”方法对 4 个小规模数据集的性能贡献非常有限，只有在数据集 Wicket 上对 F1-score 有所提高。对于大规模数据集 OpenStack，对比表 11 和表 5（最后一行）数据可以发现，所采用的“过采样”方法可以明显提高 F1-score 指标取值，尤其在模型 TextCNN 中，F1-score 指标的最大取值可以从原始数据集中的 0.410 提高到 0.681（复制 2 倍正样本情况下），即提高了 66.10%；但是，在正样本复制 3 倍的情况下，F1-score 指标的取值则呈下降趋势。此外，在 4 个小规模数据集中，“欠采样”方法对 TextCNN 和 Attention+TextRNN 模型的影响类似，所得到的性能指标值非常接近（例如，数据集 Ambari 中，在 N_{nsbr}：N_{sbr} = 10:1 的情况下，模型 TextCNN 和 Attention+TextRNN 所得的 F1-score 值都为 0.611）。而在大规模数据集 OpenStack 中，对训练样本进行“过采样”处理后，模型 TextCNN 的整体表现要优于 Attention+TextRNN。

4.3 针对RQ3的结果分析

RQ3：是否不同的深度学习模型的迭代表现相似？

深度学习采用多次迭代方式进行模型训练。在每次迭代中，根据当前读取的小批量数据样本特征和标签，通过调用反向函数 backward 计算小批量随机梯度，并调用优化算法 Adam 迭代模型参数。核心就是在无限成本取得最优到有限成本取得次优之间权衡。图 4 到图 7 是 4 个小规模数据集在不同样本均衡度情况下，F1-score 随迭代次数（Epoch）的增加而产生的变化。从图 4 到图 7 中可以看到，不同的样本均衡度下，TextCNN 随迭代的进行，F1-score 的变化幅度都要显著高于 TextRNN，这一方面受益于 CNN 的特性，另一方面也和 Dropout 层有关。并且随着迭代的增加（从 0 到 999，共 1000 次），TextCNN 和 Attention+TextRNN 均获得了一定的收益。TextCNN 的收益表现为最大值的增加，震动幅度的减小和部分图上停留在 0 处的时间减少。

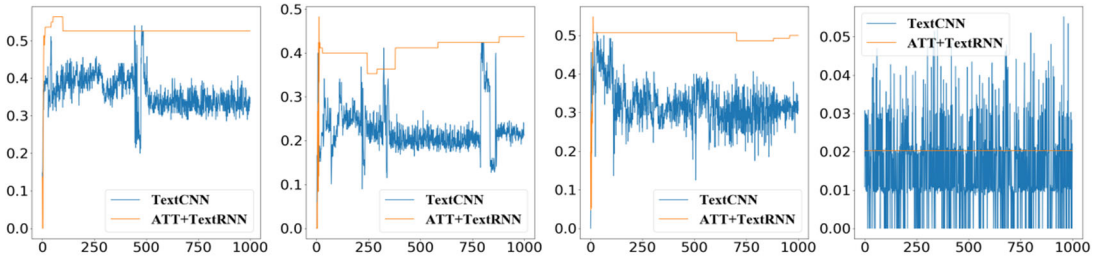
Attention+TextRNN 则是体现在稳定性的增加和最高值的增加。但是在迭代的后期，Attention+TextRNN 几乎没有收益，而 TextCNN 仍有比较明显的收益，主要表现为震动幅度的减小和部分图上停留在 0 处的时间减少。无论是 TextCNN 和 Attention+TextRNN 的最高值大部分情况下均在迭代前期达到。此外，迭代变化过程中，Attention+TextRNN 的稳定性明显优于 TextCNN。



(a)Ambari (b) Camel (c) Derby (d) Wicket

Fig. 4 The trend of F1-score by increasing Epoch (Nnsbr : Nsbr =1:1)

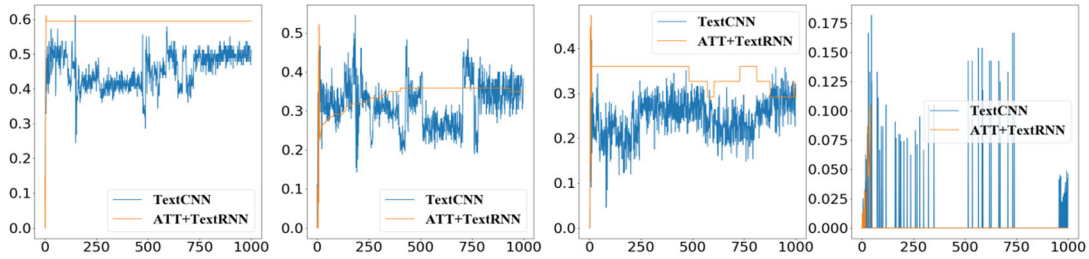
图 4 F1-score 随着迭代次数增加的变化趋势（Nnsbr : Nsbr =1:1）



(a)Ambari (b) Camel (c) Derby (d) Wicket

Fig. 5 The trend of F1-score by increasing Epoch (Nnsbr : Nsbr =5:1)

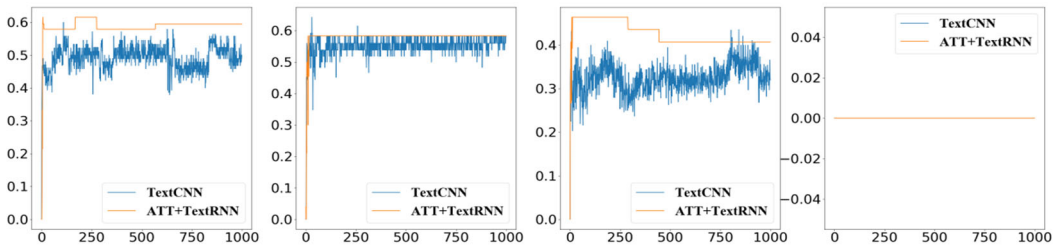
图 5 F1-score 随着迭代次数增加的变化趋势（Nnsbr : Nsbr =5:1）



(a)Ambari (b) Camel (c) Derby (d) Wicket

Fig. 6 The trend of F1-score by increasing Epoch (Nnsbr : Nsbr =10:1)

图 6 F1-score 随着迭代次数增加的变化趋势（Nnsbr : Nsbr =10:1）



(a)Ambari (b) Camel (c) Derby (d) Wicket

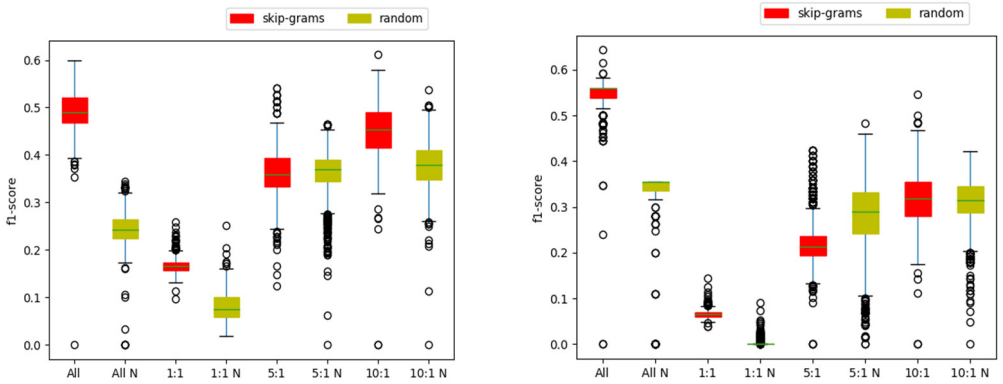
Fig. 7 The trend of F1-score by increasing Epoch (without sampling)

图 7 F1-score 随着迭代次数增加的变化趋势（未采样）

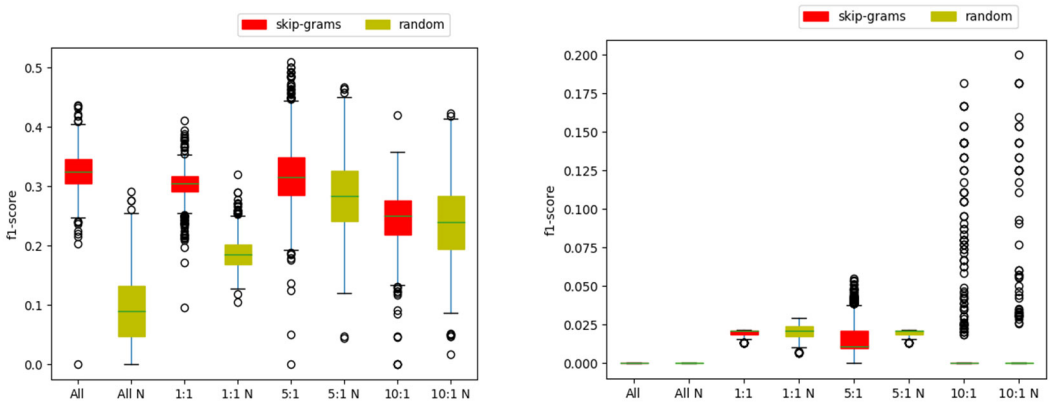
4.4 针对RQ4的结果分析

RQ4：词嵌入方法对基于深度学习的安全缺陷预测方法的性能影响程度如何？

自然语言处理中，通过词嵌入把一个维数为所有词的数量的高维空间嵌入到一个维数低得多的连续向量空间中，每个单词或词组被映射为实数域上的向量[49]。论文采用 word2vec 中的 skip-grams 方式构建词嵌入矩阵，我们将其与 PyTorch 默认的“随机生成”词向量方法进行对比，验证 skip-grams 在基于深度学习 SBR 预测中的性能影响。我们对 4 个小规模数据集在不同的采样比例下使用 word2vec 和使用随机生成词向量的 F1-score 结果进行了对比，图 8 为和图 9 分别为 TextCNN 和 Attention+TextRNN 的结果，其中标注了 N 的是随机生成词向量的结果。可以看到，使用 word2vec 在绝大多数情况下相较于随机生成词向量具有优势。因为 word2vec 生成的词向量可以很好的反映单词的相似程度，而相似程度又能辅助 TextCNN 和 Attention+TextRNN 内部参数的确定。例如如果两个没有关系的词如果具有相似的词向量，会给模型带来困扰，因为模型学习不到一个参数可以拟合这个关系。



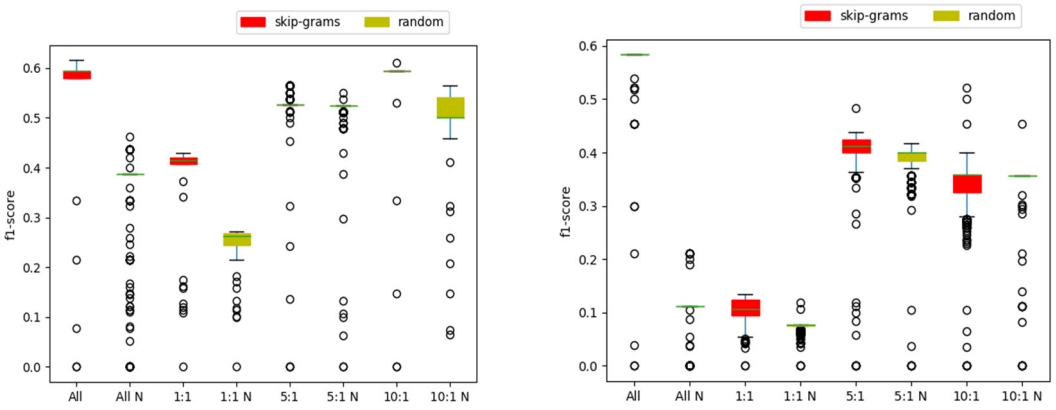
(a) Ambari (b) Camel



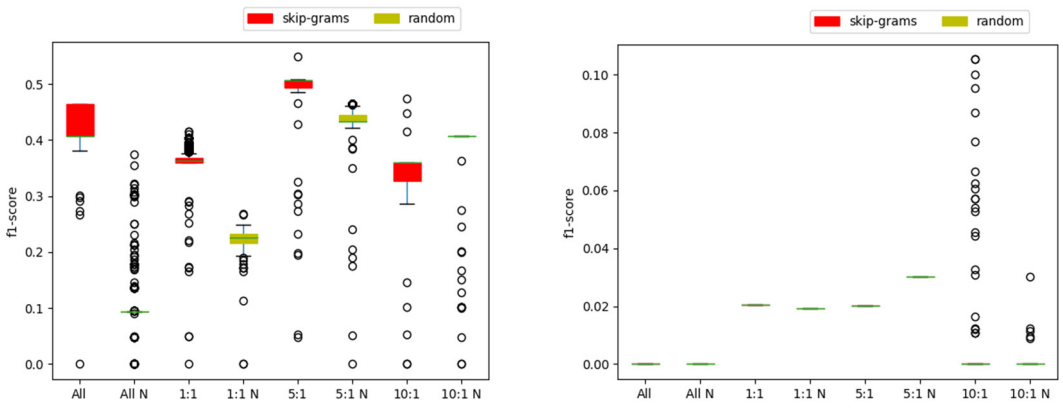
(c) Derby (d) Wicket

Fig. 8 Comparision of word2vec and Random for embedding on 4 small-scale datasets with TextCNN

图 8 4 个小规模数据集在不同采样比例下使用 vord2vec 和随机生成词向量对比（分类模型为 TextCNN）



(a) Ambari (b) Camel



(c) Derby (d) Wicket

Fig. 9 Comparision of word2vec and Random for embedding on 4 small-scale datasets with Attention + TextRNN

图 9　4 个小规模数据集在不同采样比例下使用 vord2vec 和随机生成词向量对比（分类模型为 Attention + TextRNN）

4.5 进一步讨论

4.5.1　TextCNN 模型与 Attention + TextRNN 模型的性能比较

论文设计了两种模型，即 TextCNN 和 Attention + TextRNN 模型。基于表 5 的实验结果可以看出，如果针对样本不均衡问题不进行采样处理，在数据集 Ambari、Derby 和 OpenStack 中，Attention + TextRNN 模型的性能均为最优；在 Wicket 数据集上两个模型的性能相当；在 Camel 数据集中，TextCNN 模型性能最优。结合图 7（b）可以看出，在 Camel 数据集中，TextCNN 模型只在极个别迭代时出现优于 Attention + TextRNN 模型的情况，而在大多数迭代中，依然是 Attention + TextRNN 模型的性能更优。此外，由图 7 的曲线趋势可以看出，Attention + TextRNN 模型的稳定性要显著优于 TextCNN 模型，并且，Attention + TextRNN 模型取得性能最优值的迭代次数较小，从而可以在一定程度上减少模型训练时间。在使用采样方法的情况下，TextCNN 模型在 OpenStack 数据集中的性能表现最优，通过复制正样本 2 倍，其 F1-score 取值平均可以提高 65%以上；但是，其取得性能最优值所需的迭代次数也会响应增加。

4.5.2　数据集规模和正负样本比例对模型的影响

深度学习模型对训练数据量存在极大的依赖，当训练样本过少时，会出现过拟合问题，并造成模型泛化能力较弱。论文实验所采用的四个小规模数据集总样本数量均为 1000，但是，Wicket 数据集中的正样本数量仅 10 条（仅占总样本数量的 1%），导致模型在正样本上的预测性能并不理想，在不进行采样情况下，TextCNN 模型和 Attention + TextRNN 模型的 F1-score 均为 0（如表 5 倒数第二行所示）；而在通过“欠采样”方法使得 N_{nsbr}：N_{sbr} 为 1:5 和 1:10 的情况下，其 F1-score 有小幅提升并大于 0。对于大规模数据集 OpenStack，虽然其正负样本的数量都远高于小规模数据集，但是在不进行采样的情况下，其 F1-score 取值要低于三个小规模数据集 Ambari、Camel 和 Derby。造成该问题的一个可能原因是正样本比例过低，在三个小规模数据集中，正样本比例约在 3%～9%之间，而在 OpenStack 数据集上的正样本比例仅为 0.6%。综上所述，数据集的规模以及正负样本比例对论文考虑的模型均存在较为显著的影响。

4.6 有效性影响因素分析

这一节分析可能影响实验有效性的影响因素。

内部有效性主要涉及到可能影响到实验结果正确性的内部因素。一个重要的内部影响因素就是代码的正确性。在本次实验中，为了减少人为因素，同时使我们的结论不受特殊实现的影响，使其具有普遍性，我们使用了深度学习中成熟且广泛使用的框架 PyTorch，并且使用了 PyTorch 对 CNN，LSTM 等层的默认实现。此外，我们通过相互之间进行代码评审来保证代码的质量和准确。

外部有效性主要涉及到实验的结论是否具有一般性。这里我们选用了 SBR 预测研究领域常用的 4 个公开数据集 Ambari，Camel，Derby 和 Wicket，这 4 个数据集分别来自 Apache 基金会的 4 个同名的经典项目，覆盖了多种软件的典型应用领域。除此之外，为了验证论文结论是否具有一般性，我们还额外选择了来自开源项目 OpenStack 的缺陷报告作为数据集。OpenStack 缺陷报告数据量大，来自多个项目，且我们之前的研究工作[35]已对 OpenStack 的缺陷报告进行了深入分析，为本次 SBR 预测数据集构建奠定了基础。因此可以判断实验结论具有可靠性和代表性。

结论有效性主要指实证研究中使用的评测指标是否合理。本文使用了 SBR 研究领域经常使用的多种指标：准确率、查准率、查全率和 F1-score，因此可以从多个角度对基于深度学习的 SBR 预测模型性能做出客观的评价。

5 总结和展望

本文提出了基于深度学习的 SBR 预测方法，主要使用了 TextCNN 和 Attention+TextRNN 两个模型。首先是进行了不同规模数据集中基于深度学习的方法和传统机器学习方法的性能对比，然后研究了不同的样本均衡比例对不同深度学习模型的性能影响；接着探究了不同深度学习模型的迭代表现以及不同词嵌入方法对模型 SBR 预测性能的影响。论文使用了 SBR 预测研究领域广泛使用的 4 个小规模数据集和基于开源 OpenStack 项目 BRs 构建的大规模数据集最为实证对象，结果表明，深度学习模型的预测性能要显著优于传统机器学习模型，其性能指标 F1-score 平均提升 0.258，而最好的情况下可提升 0.535。

本文属于深度学习模型在 SBR 预测研究中的初步探索，因此还存在较大的提升空间。首先在生成词向量矩阵的时候，假如训练集和测试集的词相差太多，会导致我们使用训练集测试的词向量对测试集不适用。尽管可以选择重新生成词向量，但是新生成的词向量如果套用原模型而不进行重新训练会造成模型性能下降。而如果不训练新的词向量的话词向量就会向随机词向量方向退化，同样影响模型性能。后续工作中，我们可以考虑在在一个大语料库中训练词向量，借鉴自然语言处理中的最新方法来改进词向量库。其次，由于 SBR 的类型较多（如拒绝服务攻击、SQL 注入、内存溢出等），不同类型安全缺陷特征差异性较大，论文所给出模型对不同类型的安全漏洞是否具有不同敏感性也将是我们进一步研究的方向。另外，一些新型的网络结构（例如 transformer[30]和可变卷积核[51]）是否能给 SBR 预测问题带来更好的解决方案还有待验证。

References:

[1] Amoroso, E., 2018. Recent progress in software security. IEEE Software, 2018,35(2), 11-13.

[2] CVE Website: https://www.openstack.org/

[3] CVE Detail: Websitehttps://www.cvedetails.com/vendor/11727/Openstack.html

[4] Wijayasekara, D. et al. Mining bug databases for unidentified software vulnerabilities. International Conference on Human System Interaction, HSI. November 2014 (2012), 89–96.

[5] Peters F , Tun T , Yu Y, Nuseibeh B, Text Filtering and Ranking for Security Bug Report Prediction. IEEE Transactions on Software Engineering, 2019, 45(6):615-631.

[6] Shu, R., Xia, T., Williams, L. and Menzies, T., Better Security Bug Report Classification via Hyperparameter Optimization. arXiv preprint arXiv:1905.06872.,2019.

[7] Goseva-Popstojanova, K. and Tyo, J. 2018. Identification of security related bug reports via text mining using supervised and unsupervised classification. In: Proc. of *International Conference on Software Quality, Reliability, and Security, 2018*, 344–355.

[8] Gegick M , Rotella P , Xie T . Identifying security bug reports via text mining: An industrial case study[C]// Mining Software Repositories (MSR), 2010 7th IEEE Working Conference on. IEEE, 2010.

[9] Zhang, S., Yao, L., Sun, A., & Tay, Y. (2019). Deep learning based recommender system: A survey and new perspectives. ACM Computing Surveys (CSUR), 52(1), 5.

[10] Zhang, Qingchen, Laurence T. Yang, Zhikui Chen, and Peng Li. "A survey on deep learning for big data." Information Fusion 42 (2018): 146-157.

[11] LeCun, Y. et al. Deep learning. nature 521.7553 (2015): 436.

[12] Kim, Yoon . "Convolutional Neural Networks for Sentence Classification." Eprint Arxiv， 2014.

[13] Liu P , Qiu X , Huang X . Recurrent Neural Network for Text Classification with Multi-Task Learning. 2016.

[14] Lopez, M. M., & Kalita, J. Deep Learning applied to NLP. arXiv preprint arXiv:1703.03091. 2017.

[15] Ohira, M. , Kashiwa, Y. , Yamatani, Y. , & Yoshiyuki, H. . (2015). A Dataset of High Impact Bugs: Manually-Classified Issue Reports. In: Proc. of the IEEE Working Conf. on Mining Software Repositories. 2015. 518-521.

[16] Shi, X. , Chen, Z. , Wang, H. , Yeung, D. Y. , Wong, W. K. , Woo, W. C. Convolutional lstm network: a machine learning approach for precipitation nowcasting. In: Proc. of Advances in neural information processing systems. 2015. 802-810.

[17] Devlin J, Chang M W, Lee K, et al. Bert: Pre-training of deep bidirectional transformers for language understanding[J]. arXiv preprint arXiv:1810.04805, 2018.

[18] Burges C J C . A Tutorial on Support Vector Machines for Pattern Recognition. Data Mining and Knowledge Discovery, 1998, 2(2):121-167.
[19] NLTK: http://www.nltk.org/.
[20] Yang, Z., Yang, D., Dyer, C., He, X., Smola, A. and Hovy, E., 2016. Hierarchical attention networks for document classification. In: Proc. of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies. 2016, 1480-1489.
[21] Hinton G E, Krizhevsky A, Wang S D. Transforming auto-encoders. In: Proc. of International Conference on Artificial Neural Networks. Springer, 2011, 44-51.
[22] Sabour S, Frosst N, Hinton G E. Dynamic routing between capsules. In: Proc. of Advances in neural information processing systems. 2017. 3856-3866.
[23] Yin Z , Shen Y . On the Dimensionality of Word Embedding. 2018.
[24] Mikolov T, Chen K, Corrado G, et al. Efficient estimation of word representations in vector space. arXiv preprint arXiv:1301.3781, 2013.
[25] Blei D M, Ng A Y, Jordan M I. Latent dirichlet allocation. Journal of machine Learning research, 2003, 3(Jan): 993-1022.
[26] Bojanowski P, Grave E, Joulin A, et al. Enriching word vectors with subword information. Transactions of the Association for Computational Linguistics, 2017, 5: 135-146.
[27] Pennington J, Socher R, Manning C. Glove: Global vectors for word representation. In: Proc. of the 2014 conference on empirical methods in natural language processing . 2014, 1532-1543.
[28] Peters M E, Neumann M, Iyyer M, et al. Deep contextualized word representations. arXiv preprint arXiv:1802.05365, 2018.
[29] Radford A, Narasimhan K, Salimans T. Improving language understanding by generative pre-training. URL https://s3-us-west-2. amazonaws. com/openai-assets/research-covers/languageunsupervised/language understanding paper. pdf, 2018.
[30] Vaswani A, Shazeer N, Parmar N, et al. Attention is all you need. In: Proc. of Advances in neural information processing systems. 2017: 5998-6008.
[31] Bahdanau D , Cho K , Bengio Y . Neural Machine Translation by Jointly Learning to Align and Translate. Computer Science, 2014.
[32] Krawczyk, B.,Learning from imbalanced data: open challenges and future directions. Progress in Artificial Intelligence, 5(4),221-232,2016.
[33] Huang C, Li Y, Change Loy C, Tang X. Learning deep representation for imbalanced classification. In Proc. of the IEEE conference on computer vision and pattern recognition 2016:5375-5384.
[34] Song, Jia, Xianglin Huang, Sijun Qin, and Qing Song. "A bi-directional sampling based on K-means method for imbalance text classification." In Proc. of the 15th International Conference on Computer and Information Science, 2016: 1-5.
[35] Zheng W, Feng C, Yu T, et al. Towards understanding bugs in an open source cloud management stack: An empirical study of OpenStack software bugs Journal of Systems and Software, 2019, 151: 210-223.
[36] Chaparro O, Lu J, Zampetti F, et al. Detecting missing information in bug descriptions. In: Proc. of Joint Meeting on Foundations of Software Engineering. 2017.
[37] Christopher M Bishop. Pattern recognition and machine learning. springer, 2006.
[38] Ohira M , Kashiwa Y , Yamatani Y , et al. A Dataset of High Impact Bugs: Manually-Classified Issue Reports[C]// Mining Software Repositories. IEEE, 2015.
[39] Ambari: http://ambari.apache.org/.
[40] Camel: http://camel.apache.org/.
[41] Derby: http://db.apache.org/derby/.
[42] Wicket: http://wicket.apache.org/.
[43] Apache: http://db.apache.org/.
[44] OpenStack Bugs in LaunchPad: https://bugs.launchpad.net/openstack/.
[45] LaunchPad: https://launchpad.net/.
[46] Greff K , Srivastava R K , Koutník, Jan, et al. LSTM: A Search Space Odyssey. IEEE Transactions on Neural Networks & Learning Systems, 2015, 28(10):2222-2232.

[47] Romano, J.: Appropriate statistics for ordinal level data: Should we really be using t-test and cohen'sd for evaluating group differences on the nsse and other surveys. In Proc. of the annual meeting of the Florida Association of Institutional Research.2006.

[48] Fan, Y., Xia, X., Lo, D., Hassan, A.E., Chaff from the Wheat: Characterizing and Determining Valid Bug Reports. IEEE Transactions on Software Engineering, 2018.

[49] Lai, Siwei, Kang Liu, Shizhu He, and Jun Zhao. "How to generate a good word embedding." IEEE Intelligent Systems31, no. 6 (2016): 5-14.s

[50] PyTorch official website: https://pytorch.org/.

[51] Dai, J., Qi, H., Xiong, Y., Li, Y., Zhang, G., Hu, H., Wei, Y. Deformable convolutional networks. In: Proc. of the IEEE international conference on computer vision. 2017,764-773.

[52] Bahdanau, D., Cho, K., & Bengio, Y. Neural machine translation by jointly learning to align and translate. Computer Science, 2014.

[53] Vaswani, Ashish, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. "Attention is all you need." In Advances in neural information processing systems, pp. 5998-6008. 2017.