单位代码：10293 密　级：

南京邮电大学

专 业 学 位 硕 士 论 文

论文题目：基于卷积神经网络的 issue 分类研究

学　　号	1219043803
姓　　名	徐俊辉
导　　师	张卫丰
专 业 类 别	工程硕士
类　　型	全日制
专业（领域）	软件工程
论文提交日期	2022.5.16

Research on Issue Classification Method Based on Convolutional Neural Network

Thesis Submitted to Nanjing University of Posts and Telecommunications for the Degree of

Master of Engineering

By

Xu Junhui

Supervisor: Prof. Zhang Weifeng

May 2022

摘要

issue 追踪系统作为 Github 的重要组成部分，越来越多的用户和开发者使用它提交问题报告。这些问题报告可以是项目中存在的缺陷，也可以是期待的新功能等，简称为 issue。通过提交 issue，一方面可以让开发团队更好地了解用户的需求，另一方面也能促进软件项目的迭代和完善。然而在一个项目中通常有数百个 issue，开发人员逐条审阅和处理需要消耗大量的时间和精力。Github 提供了标签机制来为加快 issue 的处理过程，标签可以让开发人员及时了解 issue 的类型信息，确定处理的优先级顺序，但是标签的使用在 Github 上依然很少，很大的原因就是标签的分配依赖人工。因此，本文着眼于自动化 issue 分类研究，为每个 issue 贴上标签，帮助开发人员提高 issue 处理效率。本论文的主要工作如下：

（1）提出一种基于词向量和主题向量的特征表示方法。issue 数据集由 issue 的标题和描述信息组成，word2vec 词向量模型可以把每个单词向量化表示，但这样的表示形式缺乏整体语义上的考量，本文在此基础上加入 LDA 主题模型，将词向量和 issue 的主题向量组合起来作为新的输入向量，特征表示更为充分，兼顾了词粒度和 issue 文本粒度。

（2）提出一种基于卷积神经网络的 issue 分类方法。注意力机制的引入让关键的特征获得更大的注意力权重，卷积神经网络的卷积层设计了不同尺寸的卷积核来提取深层语义特征，池化层采用最大池化方法进行降维和压缩，全连接层连接特征并最终通过 softmax 分类得到类别信息。实验结果表明，本文模型在准确率、召回率和 F-score 上分别达到了 81.3%、80.6%和 80.3%，能够从 issue 文本中提取到重要的特征信息实现分类，而且本文还进行了一项人工实验来说明生成的分类标签能够有效缩短研究人员审阅 issue 的时间。

关键词： issue 追踪系统，词向量，主题模型，卷积神经网络

Abstract

As an important part of Github, the issue tracking system is used by more and more users and developers to submit issue reports. These issue reports can be bugs that exist in the project, new features that users expect, and so on. They can be simply called issues. By submitting an issue, on the one hand, the development team can better understand the needs of users, and on the other hand, it can also promote the iteration and improvement of the software project. However, there are usually hundreds of issues in a project. It takes a lot of time and energy for developers to review and process them one by one. Github provides a tagging mechanism to speed up the processing of issues. Tags can let developers know the type of issue in time and determine the priority order of processing. However, the use of tags is still rare on Github. The main reason is that the allocation of tags relies on labor. Therefore, this thesis focuses on automating issue classification to label each issue and help developers improve issue processing efficiency. The main work of this thesis is as follows:

A feature representation method based on word vector and topic vector is proposed. The issue dataset consists of title and description information. The word2vec word vector model can vectorize each word, but such a representation lacks overall semantic considerations. By introducing the LDA topic model, the word vector and the topic vector of the issue are combined as a new input vector, so that the feature representation is more sufficient. It takes into account both word granularity and issues text granularity.

An issue classification method based on convolutional neural network is proposed. The introduction of the attention mechanism allows key features to gain greater attention weight. Deep semantic features are extracted by designing convolution kernels of different sizes in the convolutional layer. The pooling layer adopts the maximum pooling method for dimensionality reduction and compression. Finally the fully connected layer connects the features and obtains the category information through softmax classification. The experimental results show that the precision, recall and F-score of the model are 81.3%, 80.6% and 80.3% respectively. It shows that the model can extract important feature information from the issue text to complete the classification task. Moreover, this thesis also conducts an artificial experiment to illustrate that the generated classification labels can effectively shorten the time for researchers to review issues.

Key words: Issue tracking system, Word vector, Topic model, Convolutional neural network

目录

第一章 绪论

1.1 研究背景与意义

在软件工程开发中，开发者和用户都扮演着重要的角色，一个成功的项目离不开所有人的努力。在实际开发时,软件项目往往会经历很多次迭代，每一次的更新都需要及时听取用户的反馈[1]。用户反馈的信息对开发人员很有价值，因为用户会重点关注软件产品的缺陷，功能上的不足，体验上可以改善的地方，这些需求能够很好地帮助项目提高质量。在软件工程领域，如何对用户反馈进行收集、管理和处理值得重点关注，人工的方式显然耗时耗力。

随着版本控制系统的发展，分布式版本控制系统已经占据主流。在分布式版本控制系统中，Git 被广泛使用，并且基于 Git 开发出了很多的代码托管平台，如 Github、Gitlab、Gitee 等。这些代码托管平台上基于用户反馈的需求都相继提供了问题报告，也就是 issue。在用户使用软件过程中，他们可以通过提交问题报告来向开发者表达软件改进的建议，如果用户是外部开发者，问题报告甚至可以是对项目代码上的修改建议。随着开源氛围的流行，越来越多的用户参与到项目整个生命周期之中，问题报告已经成为开发人员和用户之间有效的沟通桥梁。

用户反馈可以统一表现为问题报告的形式，而 issue 追踪系统则可以对问题报告进行收集和管理。在项目管理过程中，往往需要保证代码是最新的，一旦发现有性能上的缺陷或者是代码正确性上存在问题，都要立刻上报处理。另一方面，维护人员必须投入尽可能少的时间和精力来解决反馈的问题报告，保持较低的软件维护成本[2]。issue 追踪系统是实现严格而有效的软件演进任务的重要手段，项目人员可以在问题追踪系统中管理和跟进每一个问题的进展。尽管 issue 跟踪系统很有用，但当开发者面对快速增长的工作量，仍然会压力巨大，甚至失去控制[3]。

不同的代码托管平台配套有不同的 issue 追踪系统，它们的特点也不尽相同。著名的 jira 追踪系统就被很多团队使用来提交问题报告，这些 issue 报告组成信息多样，包括有标题，描述，类型和其他的一些属性。不同于传统的结构化 issue 跟踪系统，GitHub 提供了一个集成的轻量级 issue 跟踪系统。issue 提交者只需要提供一个简短的文本摘要，包含一个标题和一个可选的描述信息，就可以向 Github 上托管的项目报告一个新的 issue。虽然这种简易化的报告过程降低了进入的门槛，可以吸引更多经验上并不丰富的外部代码贡献者，但它也一定程度上使得开发团队对项目的维护工作变得复杂。当有数百个甚至更多的问题被提交，它们的性

质和质量也都不尽相同[4]，在这种情况下，项目的维护人员依然会眼花缭乱，工作效率低下。项目 facebook/react 截至 2022 年 2 月 9 日已有 10241 个问题被解决，还有 714 个问题处于开放状态，这一定程度上也反映出 issue 数量之多，处理这些 issue 的工作量之大。

为了解决上述问题，Github 还另外提供了一个标签系统，开发人员可以使用标签来标记和管理报告的 issue。特别的是，标签可以很方便地反应 issue 的基本情况，比如说 issue 的内容和什么相关，主题是什么，优先级有多高等，这可以起到很好地分类和过滤的作用，有效促进项目的管理[5]，标签机制对最终 issue 是否被解决有着非常积极的影响。但与之同时，标签的分配是人工进行的，人力成本和时间开销依然比较大。事实上，据 Bissyande 等人[6]的研究显示：标签机制在 issue 追踪系统的使用率并没有很高，在收集的 803840 的问题报告中，只有 218476 个带有标签。

因此，如果可以根据实际应用中常用的标签，帮助 issue 跟踪系统区分开大量的 issue 报告，代替原来的人工标记标签，将很好地帮助项目人员加快 issue 处理过程。Cabot 等人在 2015 年完成的开源项目中 issue 标签使用情况研究发现，bug、enhancement、question 是最受欢迎的标签，占据了收集的 issue 的绝大部分，在统计的 Github 使用的主要 7 种标签中占比达到了 90%[7]。考虑到这三种标签也是 issue 追踪系统中默认的标签，本文的主要目标就是将 issue 区分成三类。我们可以认为，bug 标签表达的是开发过程中存在的缺陷或者功能上没有和预期一致，enhancement 代表对新功能的请求和需要提高的地方，question 代表对项目的疑问，比如对提交的 pull request 的问题，需要更多的信息等。当外部贡献者提交了很多的 issue，自动化标签分类将明显减少手动工作中耗费的时间和精力，促进标签的使用并大大提高项目的开发和维护效率。综上所述，基于卷积神经网络的 issue 分类方法具有重要的研究价值和实现意义。

1.2 国内外研究现状

1.2.1 围绕 Github 的各方面研究

作为目前最流行的代码托管平台，Github 上有着海量的信息可以被获取，比如代码贡献者的信息，仓库的流行程度，issue 的数量和信息，pull request 等，这些信息使得研究人员在软件工程领域可以有很多的研究方向。Guzman 等人[8]研究了 commit 信息和所使用的编程语言，代码仓库的流行程度以及其他几个因素的关联。Bowes 等人[9]区分了 issue 的用户和评论者并讨论了他们的情感表达上的差异。Zhao[10]等人为了帮助审阅人员在有限的工作时间内可

以对pull request进行更多的决策，提出了一个排序的方法来推荐可以优先被处理的pull request。

Arturo 等人[11]对 Github 进行了一项系统性的研究，探讨了 Github 对软件开发的影响，他们的统计数据为 Github 上不同研究领域发表论文的频率，相关研究使用的方法，数据集的可用性和哪些作者的影响力较大这些课题提供了有力的证据。其中比较有趣和重要的研究发现是在和 Github 相关的研究的数据来源上，普遍的选择是 Github 提供的 API 和 GHTorrent[12]，在所有发表论文中比例高达 72.57%。而研究领域主要是 pull request 和 issue 的使用，在所有论文里占据了 30.77%，所以本文对 issue 的分类研究也是目前代码仓库方面主流的方向。

1.2.2 issue 分类问题的研究现状

issue 的分类问题，很多团队已经开展了工作。Antoniol 等人[13]的研究发现 bug 跟踪系统中包含的语言信息足够把与纠错维护相关的问题与其他问题报告分开来，他们还在三个大型开源系统：Mozilla、Eclipse 和 Jboss 中对 1800 个 issue 进行了人工分类，通过一个简单的投票机制，将 issue 分为了 bug 相关和非 bug 两大类。根据人工分类的情况，使用决策树、朴素贝叶斯和逻辑回归三种有监督的机器学习方法构建了分类器，分类器采用的数据主要是问题报告的描述和摘要的内容，是用自然语言编写的非结构化文本。

Herzig 等人[14]研究了 issue 误分类的情况，他们从 5 个项目中人工检查了超过 7000 个issue，结果有33.8%的问题报告被误分类，这样的误分类情况会对issue的预测模型带来偏差，平均下来，有 39%的被标记为缺陷的文件实际上并没有存在 bug。他们制定了一组固定的规则，这些规则描述了如何根据问题报告的属性对 issue 进行分类，通过对分类结果的分析，他们认为误分类很大的原因在于开发人员和维护人员对于 bug 的理解不太一致，从而产生了偏差。

为了解决这个问题，Zhou 等人[15]不仅考虑了非结构化文本，还加入了优先级、关键字等结构化信息，通过结合文本挖掘和数据挖掘技术来预测给定的问题报告是否被分类正确，实际上误分类情况在 issue 追踪系统中一直存在。他们的方法由两个阶段组成，第一个阶段使用的是每个问题报告的总结部分，把问题报告划分成不同的等级，每个等级代表这个问题报告被正确分类的概率。分类器使用了多项朴素贝叶斯，等级包括高、中、低三种概率，输出的等级就当作是从非结构化信息抽取到的特征。接下来第二阶段再把结构化特征和第一阶段得到的特征共同作为输入，由贝叶斯网络模型来学习。

Github 上的 issue 追踪系统区别于 Zhou 等人进行实验的传统追踪系统，结构化信息并不具备。Sebastiano 等人[16]把目光转向了没有被最终修复的问题报告，识别出无法被解决的 issue

报告一方面可以让提出者及早知道他们的需求不会被满足，另一方面也能让开发人员将更多的注意力放在将被解决的实际 issue 报告上。通过文本分析并使用多个机器学习模型进行训练，最终 j48 上的表现最优，分类精度、召回率和 F-measure 均达到了 89%。此外他们的统计结果发现开发人员倾向于关闭带有 wontfix 标签的 issue 报告或者是请求的特性是没有必要的 issue 报告。

Carlos 等人[17]从 github 上选择了 7 个流行项目，通过 GHTorrent 获取到了问题的标题、描述和评论信息并保存在数据库中，使用 SVM 和 NBM 作为分类器进行实验，分类的指标是 AUROC 和 F1 值。他们工作的特别之处在于他们认为从功能的角度上看，一个问题报告可以有多个标签，于是采用了二元关联的方法来解决多标签分类，将多标签分类进行分解，转换成多个二元分类问题，每个二元分类器对应一个 issue 标签，未来他们还打算在其他项目上继续验证多标签分类器的表现。

Fan 等人[18]则是把目光放在了 issue 提交者身上，他们认为开发者如果经验丰富，那么会倾向于报告 bug 相关的 issue 并给出自己的见解。在 issue 的标题和描述信息基础上加入了开发者信息后，他们利用 SVM 进行了对比实验，实验结果是 F-score 从 0.75 指标提高到了 0.78，因此结合个人信息对于 issue 的分类可能有一定的帮助。

kallis 等人[19]考虑到 Github 上 issue 标签机制很少使用，开发了一个 Ticket Tagger 的工具，它利用标题和描述信息来自动标记问题报告，希望通过预测来推动项目里标签的广泛使用，也帮助 issue 的有效管理和优先级排序方面的研究工作。工具的核心使用了 fasttext 来作为分类器，收集了 30000 个 issue 来进行训练，并且在精度和召回率上取得了不错的效果，目前该工具已经可以集成在 Github 上新建或者已经存在的代码仓库，一旦创建新的 issue，该工具能够自动为其分配标签。

可见很多的研究一方面在分类的出发点上选择了二分类或者特定类型的识别，如 bug 和非 bug 等，虽然可以帮助识别 bug 类 issue，促进问题的早日解决，但是在粒度上不如 bug、enhancement 和 question 三分类合理，而且很多的研究并不是专注于 Github 平台的 issue；另一方面在分类方法上以传统的机器学习算法为主，本文尝试以卷积神经网络为基础来进行分类任务。

1.2.3 issue 追踪系统的已有研究

除了 issue 的分类，国内外围绕 issue 也有非常多的研究成果，比较常见的方向包括：issue 相关信息的实证研究，issue 排序，issue 生命周期预测和 issue 分类。Kikas 等人[20]认为之前对

issue 生命周期预测的研究只重视了静态特征，也就是 issue 的标题、描述、创建者等信息，在生命周期过程中，issue 还有可能被不同的人评论，这些动态信息也可能是有价值的。同时 issue 关联的项目状态也会对预测有帮助，在结合了静态、动态和上下文信息后，他们对 issue 生命周期的预测工作进行了进一步的研究和优化。

除了对 issue 标签进行探究，Bissyande 等人[6]还调研了数十万个 Github 上托管的项目，他们认为尽管已经有大量的研究提出了加强 issue 管理过程的技术，但针对 issue 使用情况，issue 追踪系统对软件项目成功的影响的调查还很有限。借助构建的数据集，他们的工作详细说明了 issue 追踪系统的使用情况，issue 的数量，issue 的提交者信息，issue 与项目成功的关系这些课题，Bissyande 的工作有力的填补了软件项目中 issue 相关研究的空白。

同 pull request 排序一样，issue 的排序研究也极具价值，毕竟太多的 issue 需要处理会对开发者的工作带来很大挑战。Dhasade 等人[21]利用 issue 追踪系统基于三个标准来对 issue 进行优先级排序：issue 的热度，issue 的生命周期和 issue 的类别。借助机器学习技术，他们开发了一个 issue prioritizer 排序器来为用户提供个性化服务，用户可以定制三个标准的权重来获取结果，未来还将继续改进机器学习模型来提高排序的效果。

安全方面的 issue 是外部人员向开发团队报告项目里的安全风险的主要手段，但现在关于这方面的研究还比较少。Noah 等人[22]致力于探究开源项目中这些问题报告的特点并且帮助开发人员改善实践活动。他们分析了 Github 上面 182 个不同项目的 3493 份安全问题报告并人工研究了其中一部分。他们发现随着时间的推移，安全上的问题报告越来越多，解决的速度也有提升，与过去几年相比，这些报告会在开发中更早的阶段被提出。但是项目人员很少参与处理这些问题报告，导致它们进展缓慢，长期悬而未决。另外一个特点就是这些 issue 提出者很少去说明他们是如何发现问题的并且也不会给出安全 issue 报告的解决办法。他们的研究成果激发了几个研究方向的需求来帮助开源社区解决安全 issue。

综上所述，issue 的分类问题已经有了大量的相关研究与成果，所有的努力都致力于提高 issue 追踪系统和 Github 等代码托管平台的繁荣发展。随着团队合作逐渐成为软件开发的主流，issue 的分类研究既利用了现有的深度学习技术来提高分类表现，又不局限于分类问题，本文包括 issue 已有的分类研究的最终目的都是加快 issue 的审阅速度，推动 issue 被早日解决，保证软件项目的维护工作，所以在论文的实验环节还加入了标签减少审阅时间的验证。

1.3 本文主要内容

本文主要应用词向量模型，主题模型和卷积神经网络技术来达到 issue 分类的目的，具体

完成了以下工作：

（1）针对数据集中存在很多无用信息的问题，本文进行了数据清洗和自然语言预处理的过程来减少嘈杂特征。

（2）针对词向量在语义信息上表达不足的问题，提出一种基于主题向量和词向量的特征表示方法。首先考虑到主题向量在整体语义表达上的优势，采用 LDA 主题模型对 issue 数据进行训练，针对不同向量维度分别进行试验，得到的主题向量反映了在不同主题上的分布情况。然后引入 word2vec 词向量模型将主题向量和词向量组合起来作为新的特征向量，新的特征向量兼顾了词粒度和 issue 整体语义，特征表示更为充分。

（3）考虑到深度学习在分类问题上的优势，提出一种基于卷积神经网络的 issue 分类方法。注意力机制来分配不同单词的权重，卷积神经网络的卷积层可以进行深层次的特征提取，池化层采用最大池化方法进行降维，全连接层拼接特征并交由 softmax 完成分类。同时划分训练集、验证集和测试集来客观评价模型，引入 dropout 等机制避免过拟合问题的出现，提升模型分类的效果。

（4）基于提出的方法，本文选取了合适的评估指标，分别在准确性和有效性上进行了验证，既从三个指标上证明了模型在分类上的表现，又说明了本文研究在减少 issue 审阅时间上的实际价值。

1.4 论文组织结构

本文围绕基于卷积神经网络的 issue 分类任务进行研究，一共分为 6 章，全文的组织结构如图 1.1 所示，具体章节内容如下：

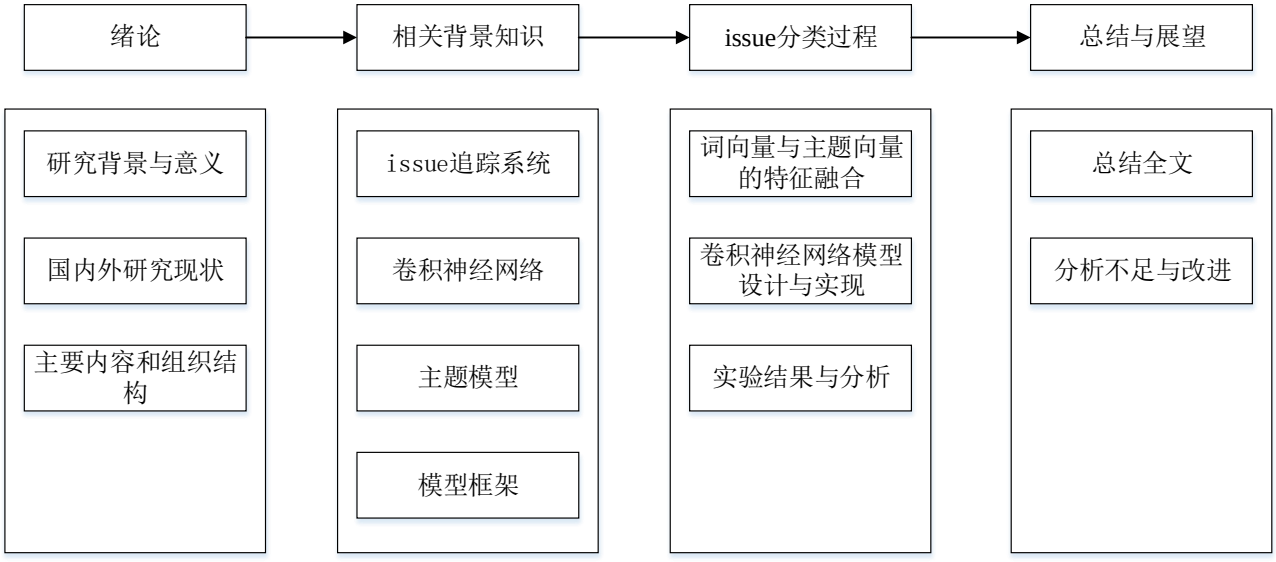

图 1.1 文章组织架构图

第一章：绪论。本章主要介绍本文研究的 issue 分类技术的背景和意义，同时还介绍了国内外在该领域的相关工作，围绕 Github 平台到 issue 分类，再到 issue 相关研究的顺序展开。此外，还介绍了本文的主要工作和论文组织结构。

第二章：相关背景知识介绍。本章介绍了本文中需要用到的技术，包括 issue 追踪系统的基本情况，主题模型，卷积神经网络，用于模型搭建的 Tensorflow 框架和 scikit-learn 机器学习库。

第三章：基于词向量和主题向量的特征融合。本章首先介绍了词向量原理和词向量训练过程，还包括了对数据集的预处理流程，预处理由数据清洗和数据处理两部分组成。接着介绍了主题模型及其训练设置，具体的还说明了利用变分推断 EM 算法求解参数的细节。最后利用词向量和主题向量实现局部和整体语义信息的融合。

第四章：基于卷积神经网络的 issue 分类。本章首先介绍了卷积神经网络框架的基础，然后详细说明了卷积神经网络的设计和实现，采用的优化方式等来实现对 issue 的预测。

第五章：实验与结果分析。对前两章提出的 issue 分类方法进行了相应的实验和对比，验证了模型的准确性和研究工作的有效性。

第六章：总结与展望。本章节主要是对本文的工作做一个总结，并介绍一下未来可以进一步研究的方向。

第二章 相关背景知识介绍

本章主要介绍本文涉及的相关知识以及技术，包括 issue 追踪系统，神经网络简介，卷积神经网络，主题模型，Tensorflow 框架和 scikit-learn 库。通过对这些相关知识和技术的介绍，为本文论文工作做好理论铺垫。

2.1 issue 追踪系统

Git 对于项目的版本控制而言是一个绝佳的选择[23]，基于 linux 开发且开源。区别于其他的版本控制工具，Git 速度很快，代码提交、修改和合并等的过程也很便捷。Github 是一个面向软件项目的代码托管平台，基于 Git 来帮助版本控制，除了提供代码管理和基本的 web 界面以外，还加入了很多辅助功能，比如 issue 追踪系统。随着现在项目复杂性的提高，基于 Github 的开发模式正逐渐成为主流。Github 的成功很大程度上来源于开发者之间相互交流与合作[2]，不管是 pull request 还是 issue，都有很大的社交属性，早在 2008 年第一版的 issue 追踪系统就已经诞生，随着不断地完善，现在默认的代码仓库都带有该功能，区别于传统的单独软件模式，Github 内置的方式使用起来更为容易。

本文的研究工作就是基于 Github 的 issue 追踪系统，它可以用来表达接下来想要完成的工作，当用户或其他开发者在使用软件产品过程中，发现项目中的 bug 需要报告，有问题向项目管理人员提问或者事先要列出准备实施的方案，都可以在 issue 追踪系统中提交一个 issue，项目开发人员讨论是否接受并在解决完成后关闭 issue，流程如下图所示：

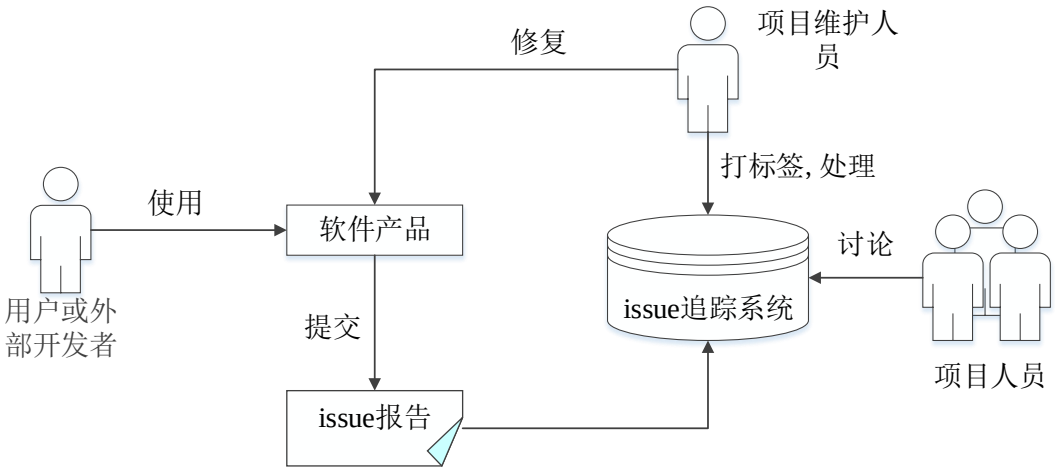

图 2.1 issue 流程图

在 Github 提供的界面上，issue 提交者只需要提交一个标题和一个简短的描述就可以创建

一个 issue。每一个 issue 被创建后都可以像图 2.1 中一样被赋予一个标签，比如是一个关于 bug 的标签等，这是一种轻量级且有用的机制，能让开发人员和外部参与人员及时了解 issue 内容，做到心中有数。开发人员可以使用一些标签来标记已发布的问题，用户通常依靠标签来快速搜索他们需要的特定主题，充分利用 issue 功能，可以提高整个项目的进度，对团队而言，这就是一个协作系统。

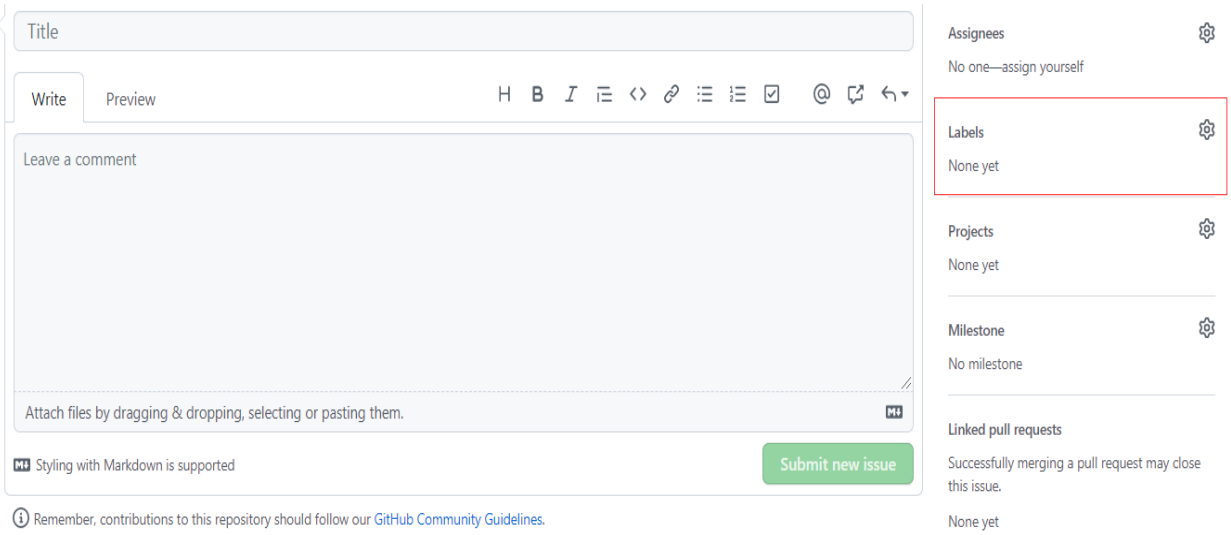

图 2.2 issue 创建图

issue 的创建可以在 Github 提供的界面完成，其效果如图 2.2，相比于其他的 issue 追踪系统，Github 的 issue 追踪系统更为的简洁，不需要像 Jira 等的 issue 一样还得分配优先级，处理人等信息，只需要提交者提交文本概述，开发人员分配标签即可。这样的方式有利有弊，一方面降低了门槛，让更多人可以参与项目，激发活跃性。另一方面也会造成很多无用的 issue 被提交，开发人员仍然需要管理标记，人力成本急剧上升，这也是 Github 上 issue 的标签没有大范围使用的原因之一。因此目前的 Github 社区也需要一种能够自动化标签的工具。

创建 issue 的步骤如下：首先通过鼠标点击 new issue，然后在页面上填写标题以及描述信息，进行 issue 创建。点击右侧的 labels 模块则可以为 issue 贴上合适的标签。创建完毕后提交 issue，等待后续项目管理人员的处理。

2.2 卷积神经网络简介

2.2.1 神经网络

人工神经网络（Artificial Neural Networks，ANN）简称神经网络，从 80 年代开始逐渐成为热点。它是一种基于生物的神经网络诞生的简单模型，可以对信息进行计算，可以用来函数进行预测。对神经网络有基本的认识和了解是学习深度学习相关模型的基础，因此接下来

会对神经网络进行一些基本的介绍。

一直以来，科学家都致力于对生物神经网络的研究，他们发现人体的神经网络使得人具有了思考的能力，神经元又是整个神经网络的基础。一般来说，人体在收到刺激后产生反应，中间都离不开神经中枢的判断，神经元是神经中枢的关键组成。于是根据神经元构建神经网络的思路得以产生，不需要像人的神经网络那么复杂，人工的神经网络只需要能够实现一部分功能即可。1943 年 McCulloch 和 Pitts 将神经元结构用一种简单模型表示，构成了一个人工神经元模型，也就是 M-P 模型[24]，如下图所示：

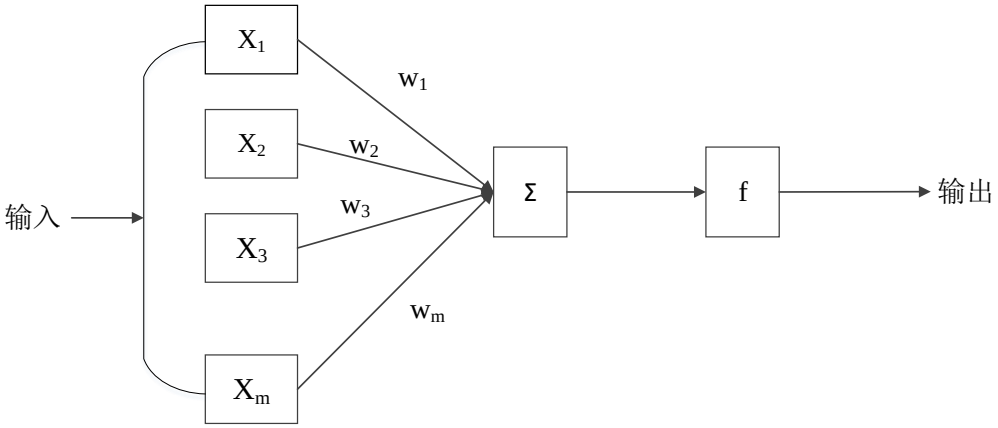

图 2.3 神经元模型图

作为最早的神经网络，MP 单层感知机网络的优缺点都很明显，优点是模型的组成结构不复杂，时间开销小，在短时间内就可以解决一些难度不大的线性问题。但是真实世界中实际问题比较复杂，单层感知器无法满足实际需求，当面对一些非线性问题时单层感知机模型已经力不从心，于是多层前馈神经网络才逐渐崭露锋芒。多层前馈神经网络不再是简单的输入层和输出层组成，隐藏层的加入能够提高神经网络计算和解决问题的能力，尤其是非线性问题。1986 年 BP 神经网络被提出，这是一种采用误差反向传播算法的多层前馈型神经网络，很好地解决了多层神经网络隐藏层之间权重的进化学习问题，目前已经是成熟可靠的神经网络[25]。

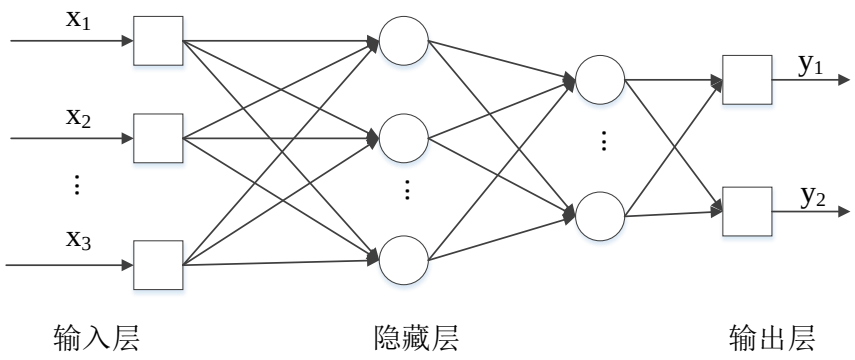

图 2.4 BP 神经网络图

图 2.4 是 BP 神经网络的结构图，可以看到隐藏层的数量没有明确要求，而且需要注意的是隐藏层的神经元数量不能太少也不能太多，如果太少，网络不能得到很好地训练，最终效果也就不好，如果太多，模型的复杂性就很高，训练时间也会延长，因此应该看实际需求来定。一般来说，在保证识别准确性的基础上，神经元数量可以少些，这样有助于避免过拟合问题。

BP 神经网络有着很多的特点：

（1）虽然 BP 神经网络有很多层，但是同一层的神经元之间没有联系。

（2）BP 神经网络的信号传递和误差反馈是相反的。信号传递是向前进行，由输入层到隐藏层再到输出层。误差的反馈则是逆向进行，从输出层到输入层，采用反向传播的方法，多次更新，以取得理想效果。

（3）BP 神经网络的传递函数是可微的，一般选择 sigmoid 函数或其他线性函数。

在训练上，神经网络一般是两种方式：

（1）有监督学习：有监督学习的每个输入相应有一个理想输出，也就是标签，训练的目的在于依靠收集到的一对对输入输出进行学习，直到达到理想状态。在训练过程中，会有偏差，这个偏差来源于实际输出和理想输出，通过迭代的方式对权值进行不断的调整，完成希望的分类任务。

（2）无监督学习：无监督学习的输入并没有对应的标签，需要让相关模型去学习输入数据的内部规律，从而能对新的数据进行识别预测。有监督学习好比是做了很多有答案的习题去参加考试，相比于直接上考场考试的无监督学习方式，自然更加轻松，不过目前也有很多任务采用的是无监督学习。

随着神经网络的快速发展，越来越多的领域开始使用神经网络来解决问题并且发展出了不同类型的神经网络，它们功能不同，各有千秋，在科研，经济，科技产品等行业发光发热，比如本文就用到了卷积神经网络模型，下一节就对其进行介绍。

2.2.2 卷积神经网络模型

受 Hubel 和 Wiesel 对猫视觉皮层电生理研究启发，Yann Lecun 提出了第一版的卷积神经网络 LeNet-5。但由于训练麻烦且效果不佳，当时的卷积神经网络并未引起很大的波澜。直到 2012 年，Hinton 引入了全新的模型结构和 dropout，在图形识别领域取得了很大的成功。作为深度学习中极具代表性的算法[25]，卷积神经网络可以在大规模的数据中提取特征并向未知数据泛化，下面对模型的结构和特点进行研究说明。

卷积神经网络模型的输入和输出具有匹配关系即可，通过学习这样大量的数据-标签集合，找到潜在规律，最终可以在输入新的数据时完成判断，一般来说数据量不要太小，否则模型效果可能不佳。卷积神经网络可以进行监督学习和无监督学习，本文而采用的是监督学习的方式，样本集的每条 issue 都由输入和预期输出组成，是一个 issue 的分类任务。

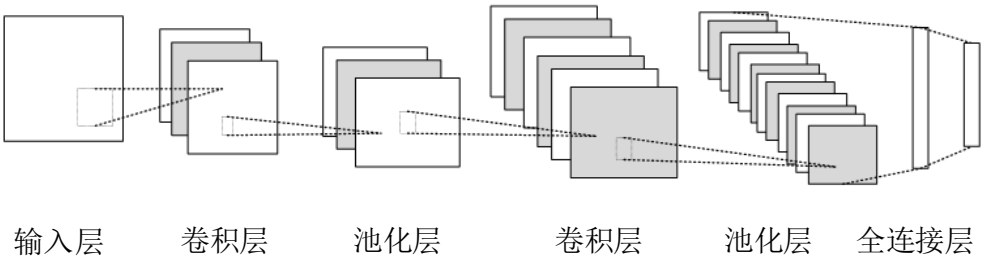

图 2.5 卷积神经网络模型图

如图 2.5 所示，基本的卷积神经网络由输入层、卷积层、池化层、输出层组成[26]。输入层主要负责读取数据，把数据表示成矢量形式，可以借助多种的词向量表示方式，最具体的形式可以是二维或者多维数组。卷积层主要进行文本或者图像等输入进行特征抽取，通常有多个卷积核，大量的计算会在该层产生，是整个模型的核心部分。在连续的卷积层间会加入池化层，能够有效减少数据和参数的量。前面的卷积和池化已经提取了非常好的特征，全连接层进行非线性组合后可得到分类结果，一般是在整个网络的末尾部分。

卷积神经网络具有很重要的两个特点：稀疏连接和权值共享。在卷积神经网络中，输入数据的维度往往很大，卷积核比起来就小很多，提取到的也就是局部的信息。这种神经元局部连接的设计一方面可以大大减少参数数量，内存消耗降低，加快学习数据的效率，另一方面可以增强网络的泛化能力，一定程度上避免过拟合的出现。图 2.6 是稀疏连接和全连接对比的示意图，可以看到稀疏链接的 n 层只是和前一层的部分相连，而全连接则是和全部神经元相连。

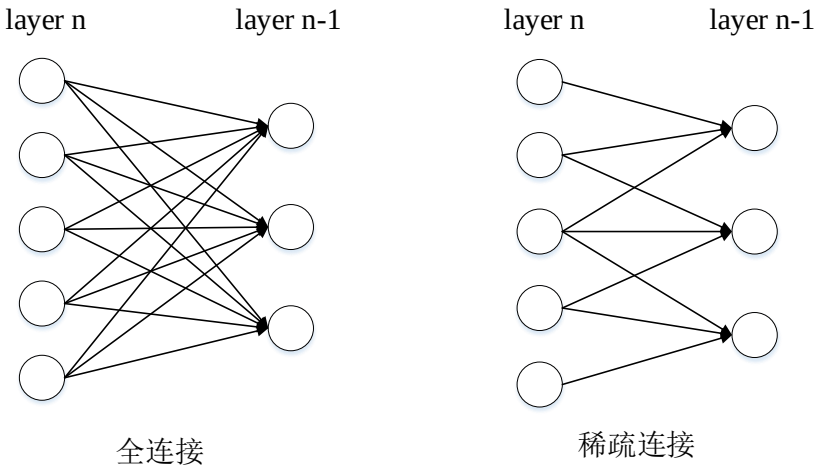

图 2.6 全连接和稀疏连接示意图

虽然稀疏链接已经显著减少了参数数量，但是如果神经元数量很多，那么参数依然不少。这个时候有了权值共享的策略，让同一层的神经元使用相同的权值参数，也就是参数复用，进一步减少参数数量的同时保证特征提取的能力。如图 2.7 所示，n 层的三个神经元共用了参数，如果一个神经元的参数是三个，那么 n 层总共的参数也只是三个，达到了目的。如果每个神经元权重参数独立，那么一共就是九个，对比明显。

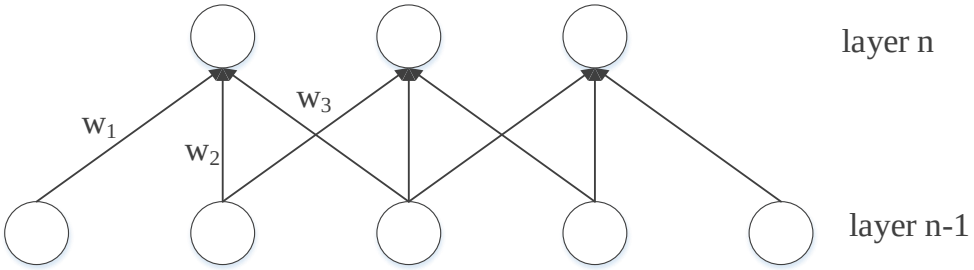

图 2.7 权值共享示意图

由于卷积神经网络上述的优点，使其在目标检测，分类等领域广泛应用，本文将在第四章详细说明模型的设计，实现和训练过程。

2.3 主题模型简介

2.3.1 LSA 主题模型

潜在语义分析(Latent Semantic Analysis，LSA)模型由 Deerwester 等人[27]提出，基于的是这样一种假设：一个单词的属性是由它所处的环境描述的。语义差不多的单词它们所在的上下文也很有可能是接近的，反之亦然。LSA 可以用文档-单词矩阵这样一种形式来反映语料库文本的情况，矩阵里的 m 行 n 列的数值就是单词出现的计数。但是原始计数的效果并不好，因为它无法体现单词在文档中的权重，于是 LSA 模型通常会用 tf-idf，即词频-逆文本频率指数来代替原始计数的方法。直观的说，一个单词在文档中出现的频率越高，权重越大，在语料库出现的频率越高，权重越小。

但是即便有了文档-单词矩阵，依然存在几个问题：

（1）文档-单词矩阵过于庞大，对计算机资源消耗很大。

（2）文档-单词矩阵含有很多噪音，包含很多无效的信息。

为了实现降维，找到近似的低阶矩阵，一般会使用奇异值分解(Singular Value Decomposition，SVD)的方法来将矩阵进行分解。如图 2.8 所示，奇异值分解可以把任意一个矩阵 M 分解成三个矩阵的乘积，即 M=U*S*V，S 是矩阵 M 奇异值的对角矩阵。很大程度上，截断 SVD 的降维方式就是选择奇异值最大的 t 个数，且只保留矩阵 U 和 V 的前 t 列，在这种

情况下，t 是一个超参数，可以根据主题数进行选择和调整。

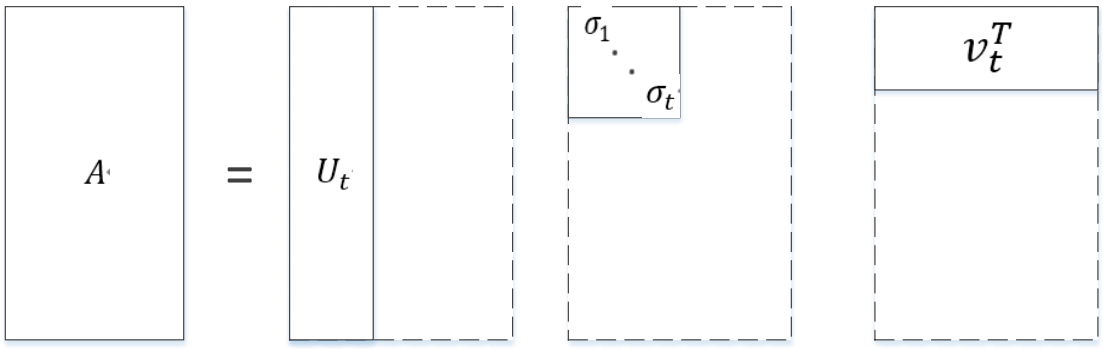

图 2.8 奇异值分解示意图

上图中的 U 和 V 就是文档-主题矩阵和主题-单词矩阵，但是分解得到的文档-主题矩阵和主题-单词矩阵并不友好，比如文档-主题矩阵，每一行为文本的主题表示，每一列代表一个主题，但是表示的数可以看作几乎没有限制的实数，这些实数难以进一步解释含义，更无法帮助从概率角度来理解这个模型。

2.3.2 PLSA 主题模型

为了克服 LSA 的弊端，Hofmann 采用概率方法来代替 SVD 并提出了概率潜在语义分析（Probability Latent Semantic Analysis，PLSA）模型[28]。模型假定文档的主题并不唯一，可能有多个主题，只是关联性上有高有低，文档的每一个词是在这些主题下以一定概率生成的。在 PLSA 里，每个文档可以看成是由两层概率分布生成的，一层是把主题看作单词上的概率分布，另一层是把文档看作主题上的概率分布。

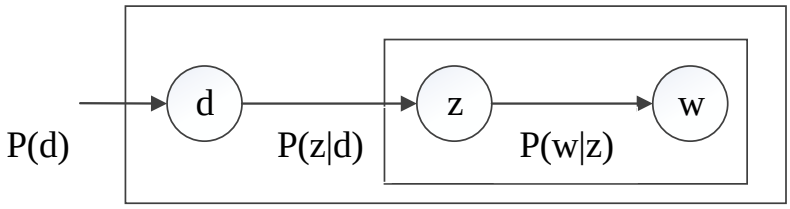

图 2.9 PLSA 模型图

如图 2.9 所示，给定文档 d，主题 z 以一定概率出现在该文档中，给定主题 z，单词 w 以一定概率出现在主题中，那么文档和单词同时出现的联合概率是：

$$P(w,d) = P(d)\sum_{z} P(z|d)P(w|z) \tag{2.1}$$

容易发现，P(d)、P(z|d)和P(w|z)是模型的三个参数，P(d)由语料库确定，P(z|d)和 P(w|z)可以使用 EM 算法训练。虽然 PLSA 和 LSA 看起来区别较大，解决问题的方式也是迥异，但其实 PLSA 也只不过是在 LSA 的基础上用概率来处理文本主题和单词。

PLSA 有了很大的进步，但依然存在一些问题，对于新的文档，由于没有参数来给P(d)建模，所以也就无法使用概率来描述。而且如果在本文使用 PLSA，随着 issue 数量的增加，模型参数的数量也会大幅度增加，造成过拟合问题，这两点也是造成 PLSA 应用不广泛的原因。

总结下，PLSA 生成文档的方式如下：

(1) 按照概率选择一篇文档。

(2) 根据选择的文档，从主题分布中按照概率选择一个隐含的主题类别。

(3) 根据选择的主题，从词分布中按照一定概率选择一个词。

2.3.3 LDA 主题模型

为了克服 PLSA 的缺点，隐含狄利克雷分配(Latent Dirichlet allocation, LDA)由 Blei 等人[29]首次提出，在主题建模等领域使用广泛。其基本思想是，每个主题都是一组基础单词的混合，而每个文档都是一组主题概率的混合。通过概率的形式，LDA 可以形象的表现文本的主题倾向和单词的主题倾向。特别的它是一种无监督学习，在训练过程中不需要人为给训练集打上标签，关注文档集和主题数量即可。

相比于 PLSA，LDA 模型是加入了参数的先验分布概念，主要对应文本下主题的概率分布和主题下单词的概率分布。这两个先验分布都是使用稀疏形式的狄利克雷分布描述的。它们可以认为是来源于生活中的一种先验思维:通常来说，一篇文章的主题只会是和少部分主题直接关联，而不会说和所有主题都有契合，是有主次之分的。主题下的单词也是如此，直观上讲，对于一个单独的主题，只有小部分词会出现比较多，大部分的词影响不大。这样两种先验使得 LDA 模型能够比 PLSA 更好地刻画文档-主题-单词这三者的关系。

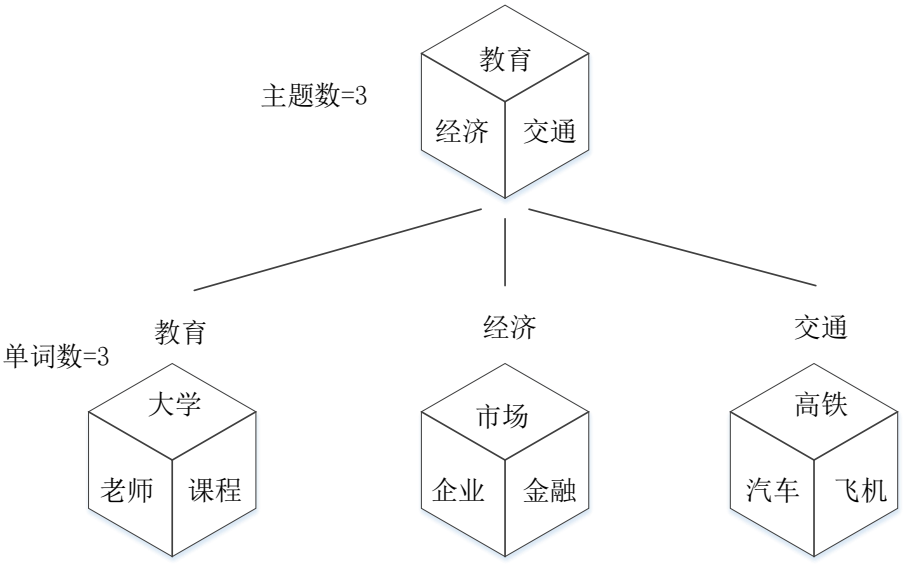

图 2.10 PLSA 和 LDA 示意图

借助上图来说明下 PLSA 和 LDA 的不同，在 PLSA 中，可以用一个概率来选取主题，根据主题，词分布也就可以确定，最后从词分布中选择一个合适的词。但是在 LDA 中，这样的过程并不适用，因为主题分布和词分布都是不确定的。LDA 中的主题分布和词分布都遵循多项式分布，而且多项式分布和狄利克雷分布是共轭的。由于加入了贝叶斯派的思想，LDA 可以认为是 PLSA 的贝叶斯化版本。

这里我们补充下 Dirichlet 分布的含义，首先引入先验分布，数据和后验分布三个概念，用公式表示就是，先验分布+数据=后验分布。用一个好人坏人例子来理解就是：根据认知，假设先验分布是 100 个好人和 100 个坏人，当遇到 2 个好人和 1 个坏人被骗时，这个即数据，就得到了 102 个好人和 101 个坏人的后验分布，而下一次过程中，这个后验分布又变成了先验分布。在数学上可以用二项分布来表示数据，Beta 分布表示先验分布，但这是二维的情况，如果说是在三维或者更多维的情况下，就对应的是多维分布和多维的 Beta 分布，一般超过二维的 Beta 分布被叫做狄利克雷分布。

基于训练好的 LDA 模型，语料库的每条语句都有主题上的概率分布，而且每个主题也有相应的高匹配词，即文本-主题分布[30]和主题-词分布。为了求解这两种分布，可以选择 Gibbs 采样算法或者变分推断 EM 算法。另外主题数的设定也是可以根据需求，50 或者 100 等都是可以的，本文也是充分了利用这一点。关于 LDA 的核心算法与使用将在第三章详细阐述。

2.4 Tensorflow 框架

Tensorflow 的发展离不开谷歌开发的 DistBelief，这是一个网络算法库，Tensorflow 就是在它基础上发展而来。作为一款开源框架，现在很多算法都以此实现，比如卷积神经网络，循环神经网络等。经过多个版本的更新，Tensorflow 使用上越来越人性化，很多不友好的功能都在逐步被解决。

在强大的生态系统的加持下，开发者可以使用各种库、工具来研究机器学习课题。而且 Tensorflow 不再是 python 使用者的专利，现在的它可以在多种语言下运行，比如 java 等。借助 tensorflow 可以很方便的搭建自己的深度学习框架，相比于其他的深度学习框架，Tensorflow 具有自己的优势：

（1）适配性强：不管是 cpu 还是 gpu，Tensorflow 都能运转。即便是家用电脑，使用者同样能完成深度学习的任务，而且在 Tensorflow 上大家可以相互分享自己的代码，取长补短。

（2）自动求微分：求微分导数是机器学习算法中常见的需求，基于 tensorflow，使用者要做的事就是自己设计好模型结构，选择合适的功能函数。根据用户输入的数据，Tensorflow

会自动求微分。

（3）实验表现出色：兼顾了效率和效果，并不会为了模型表现牺牲速度，轻松搭建需要的模型。借助 API，Tensorflow 可以应对很多复杂的需求，通过内置的可视化工具，可以实现对模型运行过程的及时查看，对模型训练过程中存在的问题也是一目了然。同时不断加入的附加库和不断完善的生态体系，都让用户愿意去使用它。

自从 2015 年发布以来，TensorFlow 在全球已有 4100 万的下载。作为全球使用广泛的机器学习框架，TensorFlow 逐渐成为了端到端的成熟平台，有着完整的生态体系。随着 TensorFlow 的日益完善，它更为简单易用，更为灵活强大，更为生产环境可用。本文所使用的卷积神经网络也是建立在 Tensorflow 之上，帮助实验能够快速开展，减轻了开发上的压力。

2.5 scikit-learn

提到 scikit-learn 就不得不说到 scipy，这是一个开源的工具包，主要为科研人员服务。随着不同领域的要求，基于 scipy 有了很多的分支，在所有分支中使用最多的就是 scikit-learn[31]，主要面向机器学习领域。在学习过程中，scikit-learn 还友好的提供了一些数据集，利用这些数据集可以方便地进行一些简单的动手实验，在实践中加深对复杂理论的理解。同时它包含有多种的聚类，回归和分类算法，包括支持向量机，随机森林，梯度提升和 K 均值等。 通过和一些 python 库的搭配，scikit-learn 成为了很多人的选择，每个人都可以访问并在不同场景下根据实际情况复用。

借助 scikit-learn，不需要花费太多的时间在编程上，代码也很简洁，可以把更多的关注放在模型以及参数的调整上，而且 scikit-learn 在监督学习和无监督学习任务中都可胜任。其中大部分的函数可分为估计器和转化器两类。估计器代表模型，可用于对数据进行预测和回归，主要有以下几个方法：

（1）fit(x,y)：训练模型，训练时间看数据集大小，参数设置等。

（2）score(x,y)：对模型的准确性进行评估，当然这不是唯一标准。

（3）predict(x)：对数据进行预测。

转化器负责对数据的处理，如降维和特征选择等，相关方法包括：

（1）transform(x,y)：对输入数据进行转化并输出。

（2）fit_transform(x,y)：在计算数据变化之前先进行就地转化。

在使用过程中，需要依据官方文档进行必要调整，本文的主题模型和预处理等过程都用到了 scikit-learn。

2.6 本章小结

本章主要介绍了 issue 分类系统涉及到的一些理论基础。本章首先介绍了 issue 系统的背景和功能上的知识，简要阐述了创建一个 issue 的流程。然后介绍了神经网络的发展和基本情况，以及本文使用的卷积神经网络的结构和重要特征。随后，介绍了三种主题模型并分析了其演变过程，讨论了优势和存在的不足。最后，介绍了 Tensorflow 框架和 scikit-learn 工具包，它们帮助搭建了本文的实验环境。

第三章 基于词向量和主题向量的特征表示

本文使用卷积神经网络来做 issue 的分类，在模型训练之前对 issue 文本的抽象语义信息抽取就至关重要。issue 的文本相对于其他的自然语言文本更加的复杂，特征的提取会影响到模型的识别效果，所以本章的主要工作就是对 issue 文本的特征表示进行优化。

3.1 动机与算法流程

输入层会把 issue 的特征数据加入到卷积神经网络中去，特征的表示方法对于后续模型的处理和分类效果有着直接的影响。对于向量化表示方法而言，一般有表层信息提取和深层信息提取两种，表层提取即 one-hot 编码，这种方式存在明显的缺陷，下节将会说明，而深层提取一般采用 word2vec 模型通过上下文单词挖掘词语含义，每个单词有单独的词向量。但是 word2vec 模型对于整体语义的表达是缺失的，LDA 主题模型正好可以弥补这个问题，通过分析语句在不同主题上的概率构建主题向量，反映了整体语义上的信息，两者想结合的方式保证了文本特征的完整性，通过词粒度和文本粒度两个层面构建词义和语义的特征矩阵。

具体来说，是要分别训练词向量模型和 LDA 主题模型，实验过程如图 3.1 所示：

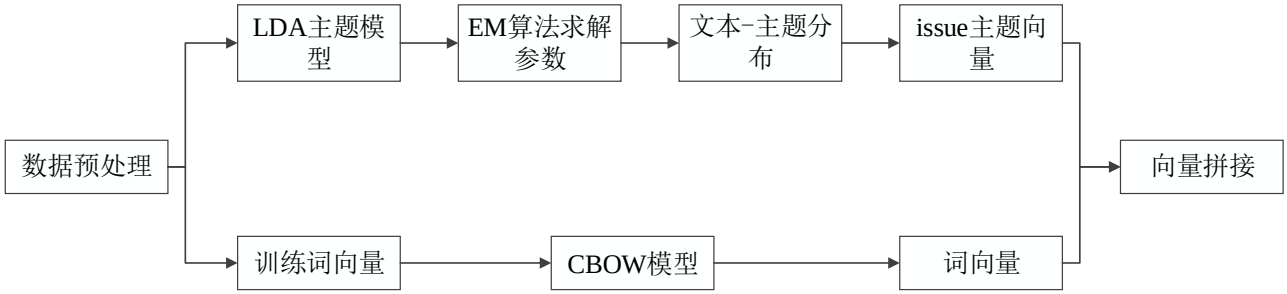

图 3.1 向量拼接示意图

本章主要内容如下：

（1）word2vec 将 issue 文本向量化，选取 CBOW 模型训练词向量，获得单词的上下文语义信息。

（2）训练 LDA 主题模型，使用变分推断 EM 算法来求解参数，得到文本的主题向量。

（3）通过特征融合的方式，结合 word2vec 词向量模型和 LDA 主题模型作为后面卷积神经网络输入层的向量表示。

3.2 词向量模型的训练

3.2.1 词向量模型原理

嵌入是通过学习句子或图像等，将其表示成矢量，并且类似的实体具有更接近的矢量[32]。issue 的标题和描述信息中的单词可以借助词向量模型表示成固定长度的向量，语义上相似的单词，它们向量的余弦距离也会越大。

传统的词向量技术是使用 One-hot 编码，这种方式虽然实现了向量表示，但没有顾及到维度的控制。词表有多大，词的向量维度就有多大，这显然存在不合理的地方。如图 3.2 所示，假设四个单词都可以用一个向量表示，每个向量中大部分都是 0，只有一位是 1，这样不仅扩展了样本的特征数量，还让分类器可以更好的处理离散数据，但是这种方式让向量过于稀疏且没有考虑到文本中的单词顺序的问题，最后的向量维度也可能很大。

Apple	[0,0,0,0,0,1,..0]
Banana	[0,0,1,0,0,0,..0]
Mango	[0,0,0,1,0,1,..0]
Lemon	[0,0,0,0,0,0,..1]

图 3.2 One-hot 编码方式示例图

为了让词向量能够实现低维稠密和蕴含语义信息的特点，本文使用 word2vec[33]来将词转化为词向量，word2vec 是一个词嵌入方法，通过训练，它可以将每个单词表示成一个 n 维的向量[34]。word2vec 具有高效的优点，短时间内可以训练大量的数据。如图 3.3 所示，word2vec 是一个简化的神经网络模型，包含输入层、隐藏层和输出层，模型输入是 One-Hot 向量，隐藏层没有激活函数，也就是线性的单元，输出维度跟输入层的维度一样。

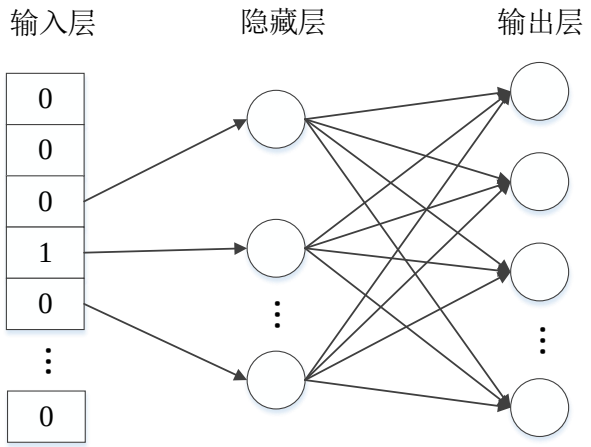

图 3.3 word2vec 模型图

相比于神经网络语言模型，word2vec 有了很多的改进之处：

（1）在模型结构上进行了优化，加快了训练速度。

（2）模型的输入方式变为了向量相加平均合成法，再一次简化了运算，代替了向量拼接的方式。

（3）相比神经网络语言模型仅采用前 n-1 个单词作为上下文，word2vec 可以选择窗口大小为 n 个词作为上下文。

（4）神经网络模型在数据集很大的情况下计算量很大，word2vec 在神经网络语言模型基础上加入了加速算法。

word2vec 词向量有两种训练模式，CBOW(Continuous Bag-of-Words Model)和 Skip-Gram (Continuous Skip-gram Model)。其中 CBOW 模型是通过上下文来预测当前值，相当于一句话中扣掉一个词，我们来猜这个词是什么，而 Skip-Gram 恰好相反，是用当前词来预测上下文。

图 3.4 是 Skip-Gram 模型图，可以看到输入层只要一个神经元，输出层有多个，隐藏层自己指定。通过 softmax 函数可以得到一个概率分布，即在给定单词情况下，其他单词出现的概率。

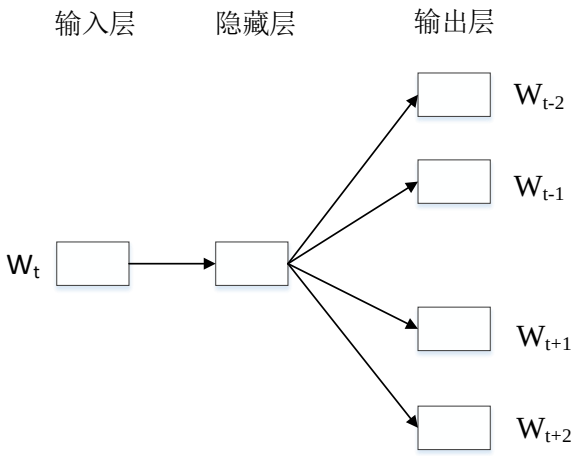

图 3.4 Skip-Gram 模型图

本文使用的是 CBOW 模型，如图 3.5 所示，是一个三层结构，输入是上下文单词，均由 One-hot 编码，所有 One-hot 分别乘以权重矩阵后相加求平均作为隐藏层向量，隐藏层向量再乘以权重矩阵得到输出层向量，经过激活函数处理得到概率分布，每一维对应一个单词[35]。假设语料(“the”,“girl”,“was”,“walking”)和语料（“the”,“boy”,“was”,“walking”），CBOW 模型会被发现 girl 和 boy 在语义上相近，因为它们具有相似的上下文。

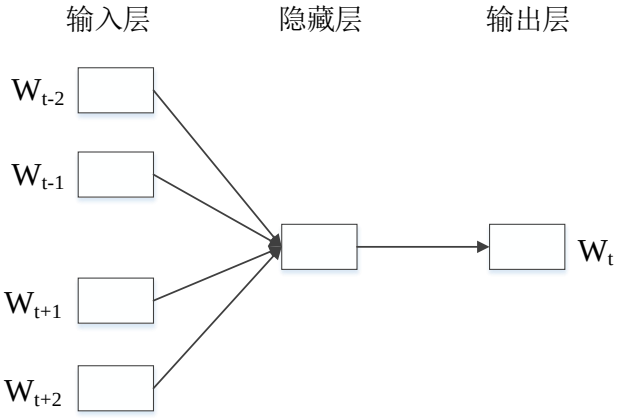

图 3.5 CBOW 模型图

3.2.2 数据预处理

在训练词向量模型和 LDA 主题模型前，需要对 issue 数据集进行必要的处理，比如去停用词等，在排除掉无关信息之后，可以得到更高质量的 issue 文本。issue 的标题和描述信息不同于普通的文本，它通常包含缩写词、代码片段、URL 以及多种语言。数据的预处理主要由两部分组成：数据清洗[36]和数据预处理。

数据清洗目的是将异常数据进行清除，具体清除了两类数据：

（1）由中文、日语、韩语等语言构成的文本：实验的模型目前是针对于英语文本进行处理，对于每一条 issue 文本，会将每一个非 ascii 字符用空格代替，具体使用了 python 中的 encode()和 decode()函数。

（2）过短或过长的文本：有的 issue 文本标题和描述信息为空或者只有 1 个单词，这类数据有用的信息太少，研究意义不大。而有的 issue 文本的描述信息过长，提交者在提交时甚至会把出现问题的整段代码放入描述信息中，这类文本在进入模型处理阶段会对内存产生很大影响，在实验过程中硬件环境无法支撑。

数据清洗完后开始具体的预处理过程：使用 python3 的 NLTK 自然语言处理包[37]来帮助处理标题和描述。首先要删除代码片段、URL、引用，这些对于 issue 的分类并没有意义，反而会干扰模型的判断。

然后将 issue 文本令牌化，也就是把标题和描述转化为一个个独立的单词，每个单词会进行词干还原，抽取单词的词干或词根，词干还原是预处理中重要的组成部分，比如 questions 会被还原成 question，相比于词形还原[38]，词干还原更为的彻底[39]，具体使用了 NLTK 的 SnowballStemmer 词干提取器。接着把每个词都小写，遇到特殊单词则进行删除。最后本文建立了一个停用词列表，有一些常见的停止词和软件方面常见的停止词，这些不必要的词在不

牺牲语句含义的情况下可以安全的忽略。在预处理完后，数据集被分成了三个 txt 文件，分别保存每个类型的 issue 文本。

表 3.1 停用词表

停用词类型	停用词
常见停用词	a，an，and，are，as，at，am，be，by，for，from，has，he，in，is，it，its，of，on，that，the，to，was，were，will，with，i，do，you
软件停用词	windows，chrome，import

3.2.3 词向量模型训练

本节主要通过 CBOW 模型来训练词向量，即通过上下文来预测当前词，具体本文使用了 gensim[40]的 word2vec API 来进行训练，训练步骤如下：

（1）CBOW 模型的输入需要首先随机地初始化一个向量矩阵。

（2）每次选择一个中心词，在中心词的前后需要有多个上下文单词。

（3）中心词会被当成是一行多维的向量，假设维度为 t，整个数据集有 n 条文本。

（4）把 n 行 t 维向量分别作为 softmax 函数的输入，在反向传播的过程中对函数的权值矩阵不断进行调整优化，在此过程中 n 行 t 维向量也是随着梯度进行更新。

（5）把调整好的 t 维向量再次放到词向量矩阵所对应的行中。

（6）重复 1-5 步骤，直到把每一个中心词都遍历过去。

word2vec 模型相关的参数设置如表 3.2 所示：

表 3.2 模型参数表

参数	参数含义及设定
sentences	预处理好的数据集
size	单词向量化的维度，设定为 150
sg	模型选择，默认是 0，代表 CBOW 模型
min_count	需要计算词向量的最小词频，设置为 5
window	词向量上下文最大距离，设置为 7

在优化算法上，word2vec 模型提供了层次哈夫曼树和负采样两种算法，它们的目的都是一样的，都是为了解决 softmax 参数太多的问题。因为数据集的单词库往往是很大的，就像本文 issue 数据集中的单词库，在进行 softmax 函数计算时会消耗大量的计算机资源，但事实上不同单词的频率差别是很大的，在计算时先判断单词是否是高频率单词更为合理，如果是高

频率单词，直接结束，不是再去判断是不是次高频单词。

层次哈夫曼树把上述的判断过程当作是一个多次的二分类问题，以哈夫曼树为基础，避免计算所有词的 softmax 概率，通过统计词频，根据频率构建哈夫曼树，如图 3.6 所示，结点内是相应权值，叶子节点数就是单词库的大小。

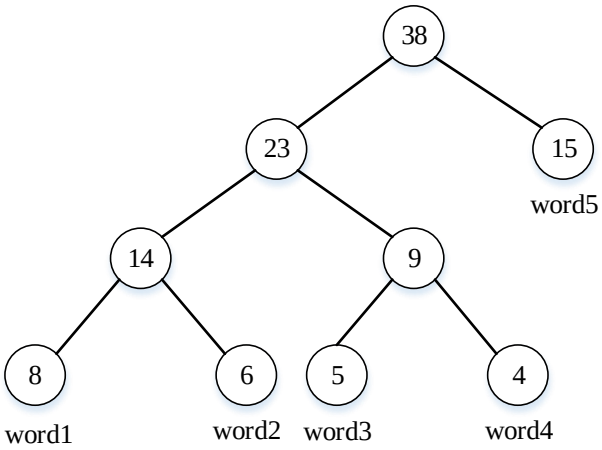

图 3.6 哈夫曼树示例图

考虑到哈夫曼树虽然可以提高训练的效率，但是如果训练样本的中心词是一个比较少见词，也就是上图中距离根节点较远的词，依然存在多次计算、效率较低的情况。所以词向量的训练还是采用了负采样的方法，假设目标词是正例，其他词就是负例，每次计算时不会把所有负例都加入，因为大部分单词和正例是关联不大的，每次选择部分负例和正例进行二元逻辑回归，从而得到每个词的模型参数和词向量。这样一方面可以提高训练速度，改善词向量质量，另一方面可以在训练过程中不必每次更新过多的权重，减少梯度下降时的计算量。

3.3 issue 主题模型的训练

3.3.1 LDA 模型的生成和训练

LDA 通过无监督学习挖掘文本中隐含的主题信息，是一种文档主题生成模型，包含词、主题和文档三层结构。第二章已经对 LDA 的基本情况进行了介绍，本节将着重对模型训练细节和参数求解进行说明。

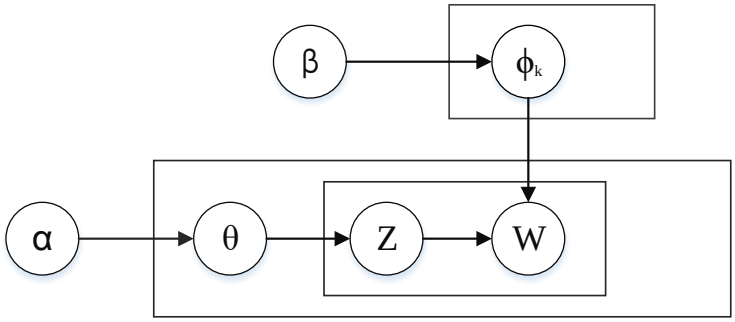

图 3.7 LDA 主题模型图

LDA 主题模型的概率图如图 3.7 所示，参数解释可见表 3.3，假设一条文本和主题满足以α为超参数的狄利克雷先验分布，主题和单词满足以β为超参数的狄利克雷先验分布。对于 issue 文本中的某个词，可以理解成首先根据超参数α选择文本-主题分布，再从文本-主题分布中选择一个主题，然后在这个主题对应的词分布中采样一个词。这样不但重复这个随机生成过程，直到一条文本全部完成。在对大规模的 issue 文本进行训练后就可以对新的 issue 进行预测，通过把 issue 文本映射到主题空间得到关于主题的分布。

表 3.3 LDA 主题模型各个参数含义表

参数	含义
α	issue 文本-主题的 Dirichlet 多项分布的先验参数
β	issue 主题-词的 Dirichlet 多项分布的先验参数
θ	issue 文本-主题分布
ϕ	主题-词分布
k	主题数
Z	issue 文本中某个词的主题
W	issue 文本的某个词

具体的本文在 LDA 主题模型的训练上主要包括 4 个步骤：

（1）加载数据集和预处理：数据集依然是 issue 的标题和描述信息构成的文本集合，在 3.2.2 节的预处理后进入步骤 2。

（2）CountVectorizer 统计词频：LDA 主题模型在训练的时候，训练数据并不是 issue 文本，而是文档-词矩阵，可以是数组或者稀疏矩阵，维度是样本数*n_features，n_features 是词的个数。因此在训练前，需要利用 CountVectorizer 来统计词频，CountVectorizer 方法的 API 可见 scikit-learn。训练好的模型可以利用 joblib 保存到路径下，方便下次直接调用。具体设置的三个参数如下表所示：

表 3.4 CountVectorizer 方法参数表

参数	含义
max_df	一个阈值，可以是 float 和 int 类型。如果是 float，代表词出现次数和语料库文档数的百分比，如果是 int，代表词的出现次数，大于该值将不会作为关键词
min_df	和 max_df 类似，如果文档频率小于 min_df，不会被作为关键词
max_features	整个语料库的词按频率排序，取前 max_features 个词

（3）LDA 主题模型训练：如果前面的处理都严格进行，模型的训练按部就班即可，对 LDA 模型的参数进行设置之后，调用 fit 方法训练数据，必要的参数设置如表 3.5，其余参数默认值即可。

表 3.5 LDA 模型参数表

参数	含义及设置
n_components	主题数，设置为了 150
max_iter	算法迭代次数，见第四点
learning_method	LDA 求解算法，设置为了 batch
evaluate_every	多久评估复杂度，设置为 200
perp_tol	复杂度容忍度 0.01

（4）迭代次数调整：迭代次数对主题模型的训练速度有很大的影响，如果迭代几十次的话，很快就可以结束，但效果不太好，在本文中迭代次数选择了 2000 次。

训练好的 LDA 主题模型可以对数据集进行主题预测，表 3.6 是选取的部分主题下的关联度最高的前 8 个主题词分布情况，这里的单词已经经过了词干提取，还原到了词根状态。

表 3.6 部分主题和主题单词图

主题	单词 1	单词 2	单词 3	单词 4	单词 5	单词 6	单词 7	单词 8
topic0	check	readm	use	common	moment	real	differ	queue
topic1	icon	attribut	abil	contract	generic	swagger	uniqu	use
topic2	descript	comment	font	usag	use	need	wallet	wiki
topic3	order	filter	databas	instanc	interact	prioriti	onlin	Payment
topic4	login	retriev	yes	stuck	cursor	nest	inline	student
topic5	section	solut	design	increas	expand	cours	hidden	term
topic6	improv	veri	cach	perform	discuss	Binf	slow	renam
topic7	store	long	dialog	calcul	avoid	role	adjust	mysql

3.3.2 基于变分推断 EM 算法的求参

在本文，LDA 主题模型的实现使用了 scikit-learn，2.5 节已经介绍了它，具体的主题模型的实现类是在 sklearn.decomposition.LatentDirichletAllocation 包。不同于 Gibbs 采样，scikit-learn 的 LDA 选择了变分推断 EM 算法[41]来得到模型最关心的文档-主题分布和主题-词分布。变分推断 EM 算法包括了变分推断和 EM 算法两部分内容。

在上节的模型图中，可以看到有隐藏变量θ，φ，z，模型的参数是α，β。为了求出模型参数和对应的隐藏变量分布，EM 算法需要在 E 步先求出隐藏变量θ，φ，z 的基于条件概率分布的期望，接着在 M 步极大化这个期望，得到更新的后验模型参数α，β，隐藏变量的概率分布如下：

$$P(\theta,\varphi,z|w,\alpha,\beta)=\frac{P(\theta\ ,\varphi,z,w|\alpha,\beta)}{P(w|\alpha,\beta)} \tag{3.1}$$

问题是在 EM 算法的 E 步，由于θ，φ，z的耦合，隐藏变量θ，φ，z 的条件概率分布难以求出，对应的期望也是如此，而变分推断可以帮助解决这个耦合的问题。变分推断的思路是假设隐藏变量是在各自的独立分布上形成的，如下图所示，耦合的变量都有一个分布对应。

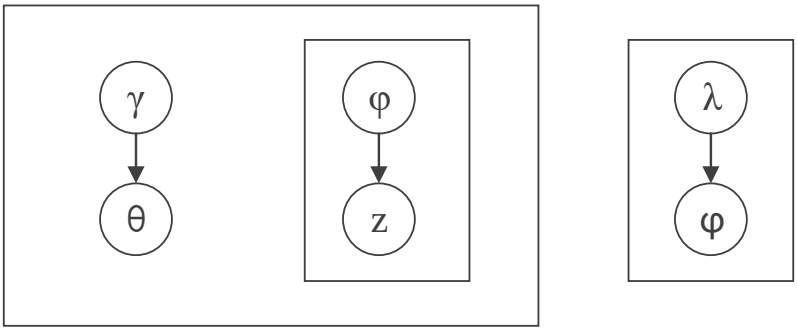

图 3.8 隐藏变量的独立分布图

假设隐藏变量θ是由独立分布γ形成的，隐藏变量z是由独立分布$\emptyset$形成的，隐藏变量φ是由独立分布λ形成的。这样就得到了三个隐藏变量联合的变分分布 q 为:

$$q(\varphi,z,\theta|\lambda,\phi,r)=\prod_{k=1}^{k}q(\varphi_k|\lambda_k)\prod_{k=1}^{k}q(\theta_d,z_d|\gamma_d,\phi_d) \tag{3.2}$$

用各个独立分布形成的变分分布来模拟近似隐藏变量的条件分布，也就是这两个分布尽可能相似，数学上讲就是 KL 距离尽可能小，这样就可以顺利的使用 EM 算法了，$D(q||p)$可以来表示 KL 距离，公式上表现为：

$$(\lambda^*,\emptyset^*,\gamma^*)=argmin\,D(q(\varphi,z,\theta|\lambda,\phi,\gamma)||p(\theta,\varphi,z|w,\alpha,\beta)) \tag{3.3}$$

找到了合适的$\lambda^*,\emptyset^*,\gamma^*$，就可以用 q 来代替 p，EM 算法也就可以使用了，而$D(q||p)$又可以用下式表示：

$$D(q||p) = \sum_{x} q(x)\, log \frac{q(x)}{p(x)} = E_{q(x)}(log\, q(x) - log\, p(x)) \tag{3.4}$$

最终$D(q||p)$函数可以转化成 ELBO(Evidence Lower Bound)函数：

$$D(q||p) = -L(\lambda, \phi, r; \alpha, \beta) + \log(w|\alpha, \beta) \tag{3.5}$$

式 3.5 中对数函数可以当作常数，最小化 KL 的距离可以转化为最大化 ELBO 函数，变分推断的问题也就是最大化 ELBO 的问题。以 ELBO 函数为基础，就可以进行 EM 算法了，通过对 ELBO 函数对各个变分参数λ，$\emptyset$，γ分别求导并令偏导数为 0，可以得到迭代表达式，多次迭代收敛后即为最佳变分参数。当变分参数收敛后，在 EM 算法的 M 步就可以固定变分参数，更新模型得到最优的参数α，β。

当进行若干轮的 E 步和 M 步的迭代更新之后，就可以得到合适的近似隐藏变量分布θ，φ，z和模型后验参数α，β，参数求解完毕，需要的 LDA 文档主题分布和主题词分布也就水到渠成。

3.4 词向量和主题向量的特征融合

词向量考虑到了上下文词间的语义信息[42]，相比于传统的特征表示方式，对语义的描述质量有了一定的提高，但是在整体语义信息上的获取却有所不足。对于 issue 的文本信息，相同类别的 issue 往往整体语义上也存在相近，或者说主题上的概率很相近。LDA 主题模型是对 issue 文本隐含语义的建模，在词向量的基础上再增加整体的主题向量，共同构成 issue 的特征矩阵，即可以丰富卷积神经网络的输入，又兼顾了词维度和语义维度，最终提高 issue 的分类效果。

利用 word2vec 的 CBOW 模型训练 issue，词向量的维度设置为 d，那么对于 issue 文本的每一个单词w_i，词向量可以表示为：

$$v_{w_i} = [v_{w_i 1} v_{w_i 2} v_{w_i 3} \dots v_{w_i d}] \tag{3.6}$$

整个 issue 文本的表示如式 3.7 所示，k 是文本的单词数：

$$v_{sentence} = \begin{bmatrix} v_{w_1 1} v_{w_1 2} v_{w_1 3} \dots v_{w_1 d} \\ v_{w_2 1} v_{w_2 2} v_{w_2 3} \dots v_{w_2 d} \\ v_{w_3 1} v_{w_3 2} v_{w_3 3} \dots v_{w_3 d} \\ \dots \\ v_{w_k 1} v_{w_k 2} v_{w_k 3} \dots v_{w_k d} \end{bmatrix} \tag{3.7}$$

训练好的 LDA 模型可以得到文本主题矩阵，主题数需要和词向量维度保持一致，对于第 i 条 issue，主题向量可以表示为：

$$z_i = [p_{i1} p_{i2} p_{i3} \dots p_{id}] \tag{3.8}$$

基于词向量和主题向量得到的 issue 最终输入矩阵为：

$$v_{sentence-i} = \begin{bmatrix} v_{w_1 1} & v_{w_1 2} & v_{w_1 3} & \cdots & v_{w_1 d} \\ v_{w_2 1} & v_{w_2 2} & v_{w_2 3} & \cdots & v_{w_2 d} \\ v_{w_3 1} & v_{w_3 2} & v_{w_3 3} & \cdots & v_{w_3 d} \\ & & \cdots & & \\ v_{w_k 1} & v_{w_k 2} & v_{w_k 3} & \cdots & v_{w_k d} \\ p_{i1} & p_{i2} & p_{i3} & \cdots & p_{id} \end{bmatrix} \tag{3.9}$$

这样的一种表示形式可以补充更多的信息，issue 文本主题向量的加入优化了卷积神经网络的输入。

3.5 本章小结

本章节中，首先通过 word2vec 训练预处理后的 issue 数据集，词向量捕捉了上下文的语义信息，然后利用 scikit-learn 构建 LDA 主题模型，得到文本-主题矩阵，每个 issue 文本可以以主题向量的形式体现在各个主题的概率情况。本文希望在词向量的基础上加入文本的主题向量，共同构成 issue 的特征矩阵，丰富整体的语义信息并作为后续卷积神经网络的输入。

第四章 基于卷积神经网络的 issue 分类

卷积神经网络可以对 issue 文本进行良好建模，第三章已经得到了丰富的语义特征，本章应用卷积神经网络来完成 issue 分类任务，并在训练上引入优化机制。所以本章主要是对卷积神经网络模型设计和实现进行介绍。

4.1 算法流程

在上一章中，实验实现了词向量和主题向量结合，构建了新的输入矩阵。本章节将使用卷积神经网络读取数据并输入融合向量进行模型训练，完成 issue 的分类任务，整体方案的流程如下图所示：

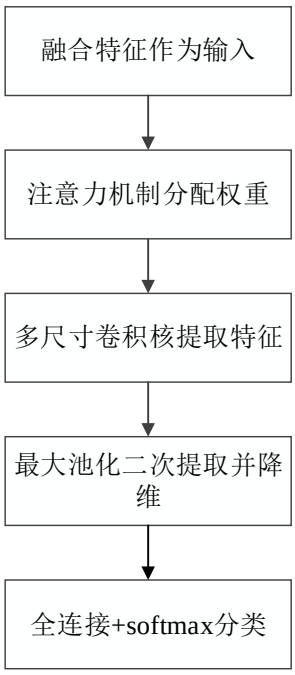

图 4.1 基于卷积神经网络的 issue 分类示意图

本章主要内容如下：

（1）介绍本文卷积神经网络涉及的基础知识，包括注意力机制、dropout 和梯度下降。考虑到采用 one-hot 编码的稀疏性，以词的嵌入表示为基础，加入主题信息，构建卷积神经网络的输入层，在输入层与卷积层之间引入注意力机制，动态分配单词权重。

（2）使用卷积神经网络特有的卷积和池化操作，设计不同尺寸的卷积核提取关键信息，最大池化方法进行降维和二次提取特征，全连接池化特征后由 softmax 进行分类，同时加入 dropout 等机制避免过拟合问题，优化模型效果。

4.2 框架基础

本节主要介绍本文卷积神经网络模型用到的一些优化机制，包括注意力机制，dropout 机制和梯度下降法。

4.2.1 注意力机制

在看一张图片时，视觉注意力会让人不会在意图片的每一个细节，而是对某些特别的点进行重点关注，对次要的像素进行过滤；在看整个书页时，人虽然可以看到所有的文字，却无法及时关注所有信息，只能先关注理解部分内容，注意力机制也是通过模拟这些特点产生。在面对序列化的信息时，注意力机制[43]会对不同的词赋予不同的权重，把有限的资源放在重要部分，使得神经网络可以灵活处理各项任务。

自提出以来，注意力机制在自然语言处理等领域帮助很多研究取得了进步[44]，在 issue 的分类问题上，注意力机制可以帮助在大量信息中聚焦关键的特征，给这些特征分配更大的权值，提升分类的效果。图 4.2 是编码器-解码器框架，现在很多注意力机制应用在该框架中。

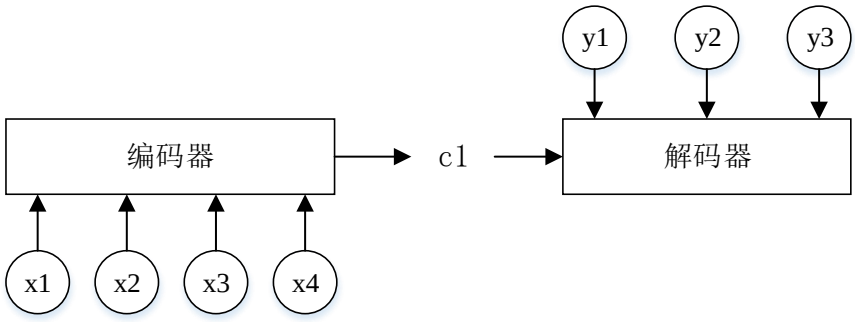

图 4.2 编码解码示例图

可以看到，语义编码 c1 维系着 Encoder 编码器和 Decoder 解码器，输入序列会经历编码和解码的过程，但由于编码会造成部分信息的丢失，编码后的输入向量就无法完整表达初始输入的含义。同时由于向量的维度是固定的，随着后面输入的不断增加，之前的输入向量的信息损失会越来越多。假设输入是 $source = \langle x_1, x_2 \dots x_m \rangle$，对应的输出是 $target = \langle y_1, y_2 \dots y_m \rangle$，输入进行一系列的非线性转换后会得到一个中间状态序列，解码器根据中间序列和已有的输出得到当前状态的输出。

在加入了注意力机制后，Decoder 解码器取消了把所有输入编码成一个固定的中间序列的方式，而是根据当前的单词生成不同状态的语义编码，如图 4.3 所示，这样也就解决了信息丢失的问题，对模型产生了差别。

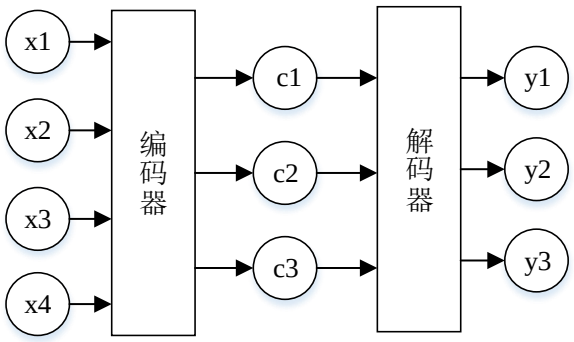

图 4.3 引入注意力机制的编码解码示例图

4.2.2 dropout

在深度学习模型中，过拟合问题很容易产生，其原因往往是：原始特征过多，混入了一些嘈杂特征，模型比较复杂等。这样情况下的模型在训练集上的分类效果很好，acc 和 loss 能保持一致的变化情况，但是在测试集上拟合效果一般，准确率会明显低于训练集的表现。

dropout[45]是深度学习中常用的解决过拟合问题的机制。dropout 以一定概率丢弃部分神经元进行训练，在网络结构更轻量的情况下，实现数据的训练。由于每次丢弃神经元都是随机的，所以在多次迭代后依然可以保证大部分神经元被训练到，还能防止参数过多造成过拟合的问题。需要注意的是，dropout 只在训练时发挥作用，在测试时不起作用，以获得最好的测试结果。

本文也使用了 dropout 机制，设置神经元停止工作的概率为 0.5，这是训练集上的设置，验证集上设置为 1.0。

4.2.3 反向传播与梯度下降

梯度下降是深度学习网络中最常用的优化方法，是一种将输出误差反馈到神经网络并自动调整参数的方法。通过计算误差值 loss 对参数的导数，并沿着导数的反方向来调节参数，经过多次的操作，将输出误差控制在最小值。直观的理解就是：为了到达山底，需要在每一步观测到最陡的地方，梯度恰好告诉了方向。梯度的方向就是所在位置上升最快的方向，那么梯度的反方向就是下降最快的方向，沿着该方向，很快就能达到局部最低点。

在对目标函数计算梯度的时候需要依靠反向传播算法。反向传播的核心是向前传播和权值更新。向前传播是在卷积神经网络模型中从输入层到输出层，逐层向前计算，直到计算出结果，再利用损失函数算出期望值和实际值之间的误差。权值更新是把向前传播中得到的误差值，沿着向前传播相反的方向逐层向后传播，更新每一个隐藏层的参数权值，以致损失函

数取到最小值。

本文采用 mini-batch 梯度下降法，将训练样本分割成一个个训练子集，在实际训练过程中每个 mini-batch 都会经历向前传播、计算损失函数、反向传播、更新参数的过程，这也叫做批处理过程，子集的大小被称为批处理的大小。相比于随机梯度下降和批梯度下降，既避免了局部最优的问题，又不用担心数据量太大影响速度和计算机性能。

4.3 卷积神经网络模型设计

CNN 最初是用在图像识别领域，图像在被转化成矩阵形式的信息后输入到卷积神经网络中去处理，这和文本的向量化过程很像，词向量模型可以把文本以矩阵形式表示，因而在文本分类上卷积神经网络也有着很多的应用[46]。卷积神经网络的最大特点是卷积和池化操作，提取关键信息的同时减少了参数数量。本文基于的是 TextCNN 分类模型，该模型使用卷积神经网络进行分类任务。相比于传统的卷积神经网络，TextCNN 可以设计不同尺寸的卷积核来捕捉 issue 文本的局部特征，其基础结构如下图所示：

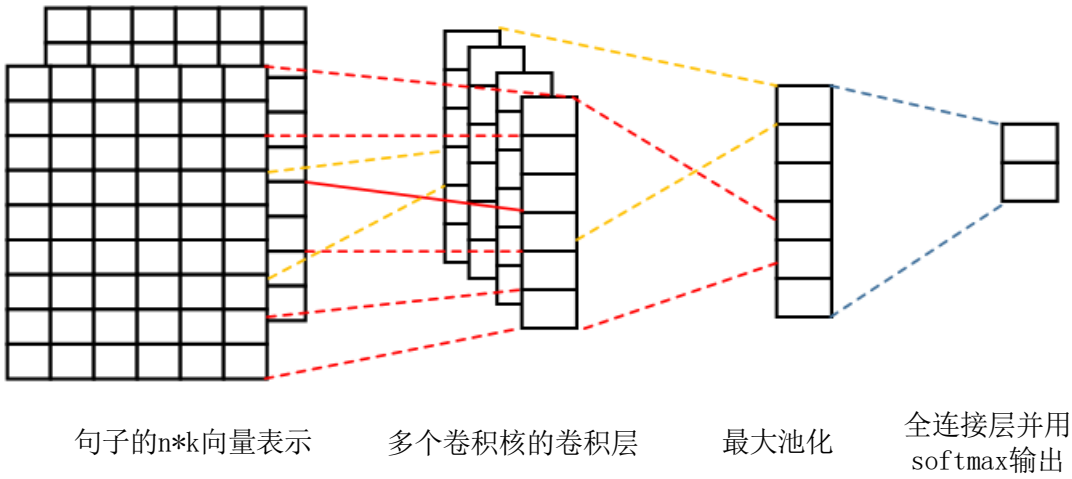

图 4.4 TextCNN 网络基础架构图

接下来将对本文卷积神经网络各层的设计进行说明：

（1）输入层设计

卷积神经网络接收二维矩阵组成的数据，但初始的 issue 数据集中的文本并不满足需求，需要先把单词转化成一定维度的向量，再把一条 issue 文本转化成长度乘以向量维度的形式。具体采用第三章中所述的词向量和 issue 文本主题向量拼接的方式，构建起输入层的输入矩阵，在保持词义特征的同时，增加整体的语义特征，矩阵表示情况如下：

$$input = \begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1n} \\ w_{21} & w_{22} & \cdots & w_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ w_{k1} & w_{k2} & \cdots & w_{kn} \\ z_{11} & z_{12} & \cdots & z_{1n} \end{bmatrix} \tag{4.1}$$

其中 w 表示 issue 每个文本的词向量表示，z 是 issue 文本对应的主题向量。

（2）注意力层设计

在实际分类过程中，不同的单词对 issue 分类的贡献是不一样的，如 error、suggest 等词语与分类类型的关联度较高，而像 class、conclusion 等单词的关联性就比较低，所以有必要把注意力集中在重要的单词上。对于注意力机制加入的位置，本文选择在输入层之后，卷积层之前，主要的考量是为了对模型的融合向量进行选择性提取，减少噪声影响。

具体来说，注意力层由两部分组成：权重计算和加权处理。权重计算是输入特征经过 softmax 激活函数计算出注意力权重，得到的注意力权重是一维向量，加权处理是将注意力权重矩阵和输入相乘得到加权的输入。在注意力资源分配到更有效的信息上并生成经过注意力权重计算的新的特征表示后，进入到卷积层的处理过程。

（3）卷积层设计

卷积层是卷积神经网络中最为重要的一层，它主要的作用是对输入层的输入数据进行特征提取，特征降维，输出特征图[47]。如图 4.5 所示，如果输入的二维矩阵是 5×5 的形式,卷积核为 3×3 的二维矩阵，步长为 1，那么通过卷积操作就可以得到 3×3 的特征图。

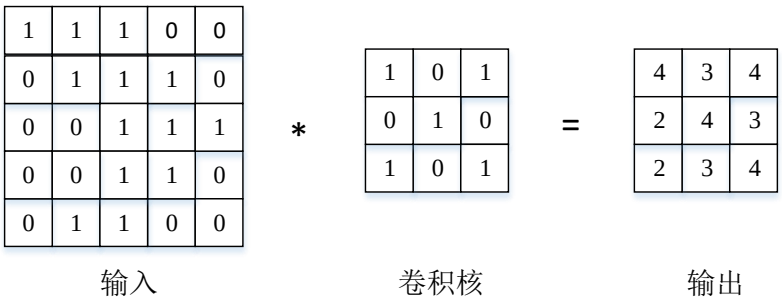

图 4.5 卷积层示意图

在本文中，为了尽可能地获取不同粒度大小的局部语义信息，卷积层使用了卷积窗口为 3，4，5 的三种卷积核，每一种卷积核会对输入数据进行卷积操作，假设窗口大小为 h 的卷积核，其中的单词卷积后输出的特征为：

$$s = f(w * a_{i:i+h-1} + b) \tag{4.2}$$

其中 f 为激活函数，w 是滤波器，a 是局部特征矩阵，b 为偏置项。

激活函数选择了目前最常用的 relu 函数[48]，函数表达式如式 4.3 所示，相比于 sigmoid 函数和 tanh 函数，relu 函数在 x>0 时，梯度一直为 1，有利于克服梯度消失的问题，收敛速度

快。relu 函数还有提高稀疏性的优点，x<0 时输出为 0，即训练完成后为 0 的神经元越多，提取出来的特征越具有代表性，泛化能力越强。

$$y = \begin{cases} x, x \geq 0 \\ 0, x < 0 \end{cases} \tag{4.3}$$

（4）池化层设计

池化层在 CNN 中也占据了很重要的位置，由于在卷积层中得到的特征图维度在通常情况下会比较大，且有很多的冗余信息，经过池化层的处理，可以抽取最关键的特征，减少下一层的参数，也起到了缩减矩阵大小的作用。同时池化层还能加快运算，一定程度上避免过拟合的问题。池化方法主要有最大池化[49]和平均池化，最大池化是取窗口里的最大值，平均池化是取所有值的平均值。在本文所提出的卷积神经网络模型中，池化层也是采用最大池化操作，找出特征图中的最大值，即影响 issue 分类结果最大的因素，并将其提取出来。如图 4.6 所示，如果卷积层的输出是一个 4×4 的特征图，池化窗口为 2×2，步长为 1，那么最大池化后就可以得到 3×3 的矩阵。

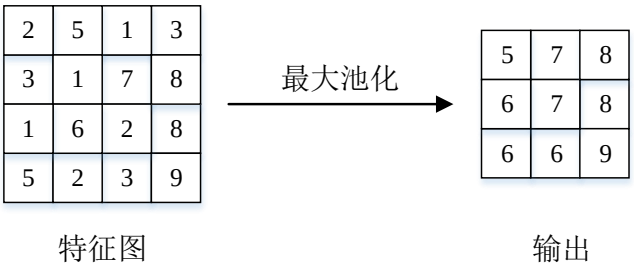

图 4.6 池化层示意图

（5）全连接层设计和分类

全连接层会把池化后的特征进行连接，形成一个综合的特征向量。对于一条 issue 语句，softmax 函数会计算全连接的输入得到 issue 属于 bug、enhancement 和 question 三种类别的概率，从整体上考虑提取的特征来完成分类任务，最后用 argmax 函数选出概率最大的作为预测结果，计算公式如 4.4 和 4.5 所示：

$$p = \text{softmax}(w^0 h^* + b^0) \tag{4.4}$$

$$y = argmax(p) \tag{4.5}$$

其中w^0和b^0待学习参数。

（6）损失函数

损失函数通过计算实际值和预测值的差别来判断模型在样本上的表现，在训练过程中也是依据损失函数的情况来更新参数。本文的误差函数选择了常见的交叉熵损失函数，公式如下：

$$H(p,q) = -\sum_{x} P(x)\, log\big(q(x)\big) \tag{4.6}$$

其中$p(x)$代表真实的概率，$q(x)$代表输出层由 softmax 的输出。

4.4 卷积神经网络模型实现

本节主要介绍卷积神经网络的具体实现，包括输入层、注意力层、卷积层、池化层、全连接层以及训练细节，整体基于 2.4 节介绍的 Tensorflow 框架搭建，同时本节还给出了具体的参数设置，一些方法的必要介绍。

（1）输入层实现

输入层会将 issue 文本以向量形式表示，每条 issue 的单词的向量都会放入矩阵中，同时会交由已经训练好的 LDA 主题模型进行识别，将处理好的 issue 主题向量也加入向量矩阵中。如果用 k 维向量表示 n 个单词组成的 issue 文本，那么可以构建出一个 n*k 的二维矩阵，由于每个 issue 的单词数量是不固定的，当有 issue 达不到最大长度时，会进行补零操作，具体本文的最大长度设置为了 256。

（2）注意力层实现

注意力机制在输入层后单独构成一层，把之前的输入矩阵记作H，$H = [h_1, h_2, h_3, \dots h_i]$，$i$是句子长度，那么注意力层的工作主要可由下面三式来表示：

$$m = \tanh(H) \tag{4.7}$$

$$alpha = \mathrm{softmax}(w^T m) \tag{4.8}$$

$$p = h * alpha \tag{4.9}$$

其中w是待学习参数，$alpha$是注意力权重，p是加权后的h值，也就是注意力概率值和对应的向量相乘。

（3）卷积层实现

借助 Tensorflow 里面实现卷积操作的 tf.nn.conv2d 函数来构建卷积层，卷积层涉及到的一些参数包括：filter_sizes 是卷积核尺寸，num_filters 是卷积核的长度，strides 表示窗口在每个维度上滑动的步长，padding 表示卷积的形式，有 VALID 或 SAME 可选，SAME 是对矩阵进行补零，VALID 是进行舍弃，经过卷积层的处理就可以得到输入数据的特征图了。

表 4.1 卷积层相关参数表

变量名	变量值
filter_sizes	3，4，5
num_filters	128
strides	[1，1，1，1]
padding	VALID

（4）池化层实现

利用 tf.nn.max_pool 方法实现池化操作，tf.nn.max_pool 主要有 4 个参数：value 是池化层的输入，一般池化层跟在卷积层后面，所以通常是特征图，参数 ksize 代表池化窗口的大小，strides、padding 前面已经介绍过。把最大池化的运算结果拼接起来得到特征向量，这样池化特征提取的工作就基本结束。具体值设置见下表：

表 4.2 池化层参数表

变量名	变量值
ksize	[1，sequence_length - filter_size + 1，1，1]
strides	[1，1，1，1]
padding	VALID

（5）全连接层和分类实现

首先使用了 tf.nn.dropout 引入了 dropout 机制。参数 dropout_keep_prob 在训练时设为 0.5，在预测时设置为 1。然后初始化矩阵 W 和 b，这两个矩阵是待训练的参数，会作为 tf.nn.xw_plus_b 方法的参数来计算 scores，具体该方法的主要参数含义如表 4.3 所示，最后根据 argmax 函数计算 scores 得到预测结果，预测的值代表属于某一类的概率。

表 4.3 tf.nn.xw_plus_b 方法参数含义表

参数名	参数含义
X	输入的矩阵
weights	矩阵的权重
biases	矩阵的偏置

（6）模型训练

模型训练中的参数主要有 batch_size，evaluate_every，checkpoint_every，num_checkpoints。由于训练集中会有很多的样本，为了提高速度，会将训练集按照 batch_size 划分若干组，evaluate_every 是间隔多少步后在验证集进行评估模型的表现，包括 acc 准确率和 loss 损失率，checkpoint 是保存点，保存着模型各个变量的值，checkpoint_every 是多久保存一次

checkpoint，num_checkpoints 就是保存的数量了，值设置如表 4.4 所示：

表 4.4 CNN 模型训练参数表

参数名	参数值
batch_size	64
evaluate_every	100
num_checkpoints	5
checkpoint_every	100

在卷积神经网络模型训练完后，会对 issue 数据进行预测，预测时调用已经训练好的模型，把预测结果保存在文件中并计算准确率、召回率和 F-score，具体指标说明将在下一章进行介绍。

4.5 本章小结

本章首先对整体的方法流程进行了介绍，接着重点对提出的 issue 分类模型进行介绍，包括卷积神经网络模型的设计和具体实现。输入层主要是第三章构建的融合向量，注意力层对不同的词赋予不同权重，卷积层使用多个卷积核提取特征，池化层抽取出 issue 的深层语义信息，全连接层连接特征并由 softmax 函数计算各类别概率，同时还对 dropout 机制，反向传播算法等进行了说明。总体上讲本文的方法适合于 issue 的分类任务。

第五章 实验与结果分析

本章对第三章和第四章提出的应用词向量模型，主题模型以及卷积神经网络的 issue 分类方法进行了实验，并对实验结果进行了分析。首先介绍了实验环境和评估指标，接着从两个角度对实验结果进行分析，分别是：issue 分类的准确性分析并和其他方法进行对比，帮助开发者审阅 issue 的有效性分析。

5.1 实验环境

表 5.1 实验环境

开发工具	配置情况
操作系统	Windows10
开发语言	Python
开发平台	IntelliJ IDEA
CPU	Intel(R) Core(TM) i5-9300H CPU @ 2.40GHz
内存	16G
硬盘	500G

本实验是基于 Python 语言，Python 版本是 3.6，开发工具使用的是 IntelliJ IDEA。实验使用 Tensorflow[50]搭建了卷积神经网络，另外还使用了第三方 Python 工具包 Gensim 来训练词向量，使用了 scikit-learn 库训练 LDA 主题模型。

5.2 实验过程

5.2.1 实验步骤和评估

本文首先对初始的数据集进行处理，包括筛选数据，预处理，词向量模型的训练和向量化过程，数据集是由 bug、enhancement、question 三个类别构成，bug 类型的文本有 10000 条，enhancement 类型 10000 条，question 类型的文本有 10000 条。然后把所有的文本交给主题模型，先进行 LDA 模型的训练，必要参数的设置和调整，为了保证模型效果，迭代次数选择了 2000 次，对于每个 issue 文本调用训练好的模型便可以得到在不同主题上的概率分布，主题向量和词向量共同形成新的特征。有了融合的特征向量后，根据构建并且训练好的卷积神经

网络模型实现 issue 的分类，并将分类结果保存在文本文件中，方便后续统计和保存。具体在实验时数据集会被划分为 10 份，选择其中 9 份作为训练集，1 份作为测试集。

通过以上几个步骤完成 issue 的分类之后，为了评判分类结果的合理性，根据已有的经验提出了两个问题进行研究：

- RQ1：issue 分类的准确性如何？
- RQ2：issue 分类的有效性如何？

5.2.2 评估指标

在评价本文对 issue 分类效果中使用精确率(precision)、召回率(recall)和 F-score 作为评价指标，这三个指标是分类任务中用来衡量性能的普遍选择。

精确率反应的是预测实例的精确程度，即预测为正的样本中有多少是真正的正样本，公式如下：

$$precision = \frac{TP}{TP + FP} \tag{5.1}$$

召回率反应了模型的敏感程度，即正确的样本中有多少被预测为正确，公式如下：

$$recall = \frac{TP}{TP + FN} \tag{5.2}$$

F-score 是精确率和召回率两者的调和平均数，用来衡量 issue 分类的完备性，公式如下：

$$F - score = \frac{2 * precision * recall}{precision + recall} \tag{5.3}$$

其中 TP 表示分类结果是正确的，样本原本属于正样本，最终也被分到了正类；FN 表示分类结果是错误的，某样本原本是正类但预测时被分成了负类；FP 表示分类结果是错误的，某样本属于负类，预测时被分成了正类；TN 表示分类结果是正确的，某样本实际上属于负类也确实被分成了负类。参数含义可由下表展示：

表 5.2 issue 分类指标参数含义表

	实际为正样本	实际为负样本
预测为正样本	TP	FP
预测为负样本	FN	TN

5.3 实验结果与分析

5.3.1 issue 分类的准确性分析

本文第三章把词向量和主题向量放一起考虑，构建新的 issue 特征。词向量着重上下文的词义特征，issue 的文本主题向量反映全局的语义特征。本文希望对于一条 issue 文本，加入一个全局主题向量可以提高分类模型的效果。

词向量和主题向量需要保持一致，不同维度的分类情况也不一样，有好有坏。为了测试不同维度词向量和文本主题向量对 issue 分类的影响，分别设定了 50 维、100 维、150 维、200 维四组实验，使用 F-score 作为了度量。实验结果如图 5.1 所示：词向量和文本主题向量的维度设定为 50 维时，效果略差于其他维度，100 维，150 维和 200 维的分类效果较为相近，其中 150 维最佳，当维度是 200 时出现了下降，说明词嵌入维度过大反而不利于单词的特征表达，在提高了时间复杂度的同时模型也会更复杂。既然向量维度确实会影响到分类精度，考虑到实验环境情况，最终使用 150 维的向量维度作为卷积神经网络的输入。

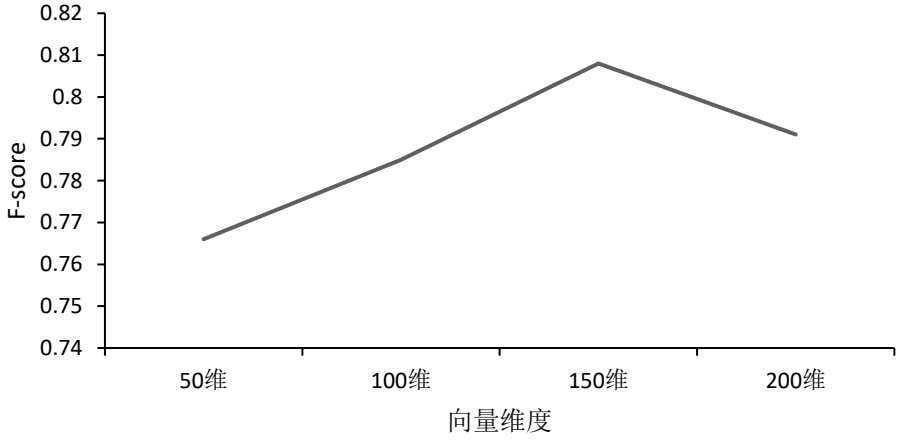

图 5.1 不同维度 F-score 情况图

本文的 issue 分类方法基于的是 Github 平台，上面的 issue 一般拥有标题，描述和评论信息。考虑到分类的及时性需求，数据集的每一条 issue 并不包含评论信息，因为 issue 在提交之后，可能过了很久才会有用户进行评论，等用户和项目人员讨论完才开始分类显然不太合适。因此在这样的数据集基础上一方面对自己的方法进行测试，另一方面也要和其他分类方法对比，本文对比的实验主要考虑了两个：

（1）基于 FastText 的 issue 分类。

（2）仅用 word2vec 进行向量表示的基于卷积神经网络的 issue 分类。

之所以选择这两种分类方法进行对比，是因为 FastText 学习速度比较快，是一个快速的文本分类器[51]，能够提供高效的表征学习能力，分类效果媲美深度神经网络。另外本文结合了词向量和主题向量，通过和仅用 word2vec 向量化的分类方法对比，也能看出主题向量对 issue 分类有无帮助。

本文提出的方法的分类表现如图 5.2 所示，在不同类别上均取得了不错的结果，分类的准确率达到了 80%以上，整体来看精确率、召回率和 F-score 三个指标上分别达到了 80.8%，80.7%，80.7%。

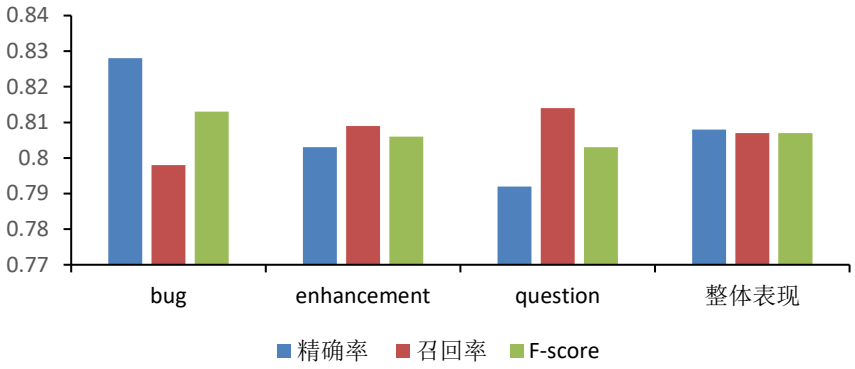

图 5.2 issue 分类效果图

图 5.2 以柱状图的方式直观展现了在不同 issue 类别上的分类效果，总体上讲实验的方法达到了预期。由于 issue 的文本较为复杂，嘈杂特征多余普通的文本，所以分类表现上不如一些传统的文本数据集，后期考虑通过进一步增加数据集并调整模型结构等措施来优化提升 issue 分类效果。

基于 FastText 的 issue 分类方法结果如下图所示，可见本文的方法是有效的，三个指标都有了一定程度的提高。具体来说精确率指标提高了 1.7%，召回率指标提高了 2.5%，F-score 指标提高了 2.1%，这证实了本文的方法能够很好地学习到 issue 的关键信息。

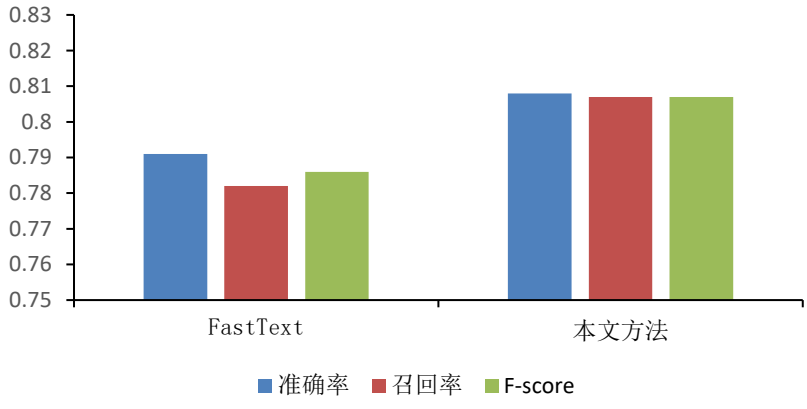

图 5.3 FastText 分类和本文方法分类对比图

同时还在单纯的使用 word2vec 词向量上进行了多次实验，不考虑 LDA 主题模型带来的

主题向量的影响，只用了词的上下文的语义信息。和本文方法的对比如图 5.4 所示，可以看出整体来说精确率、召回率、F-score 分别低了 1.6%、1.7%和 1.6%。这说明单一的 word2vec 词向量没有考虑到整体的语义信息，综合表现是不如融合的特征向量的，LDA 主题信息的加入能够有效提高 issue 分类的效果，在后续卷积神经网络的卷积层、池化层、注意力层优化调整后能够全面表示 issue 信息特征。

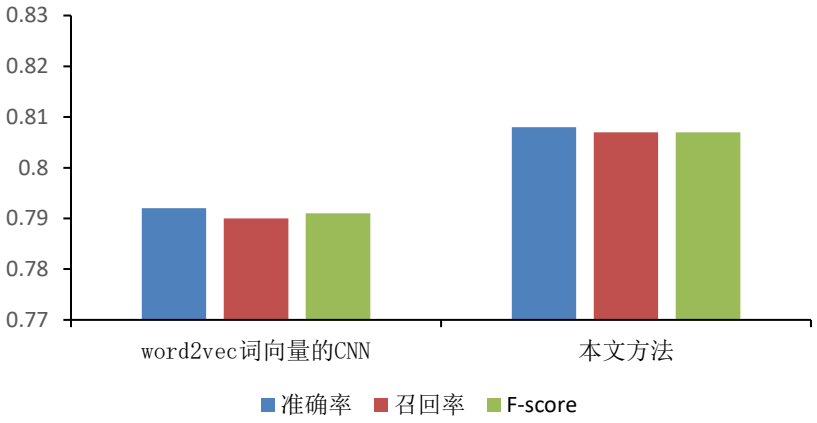

图 5.4 word2vec 词向量的 CNN 和本文方法分类对比图

所以在准确性上本文方法优于 FastText 和 word2vec+CNN 的方法，但是时间开销上更大，FastText 在训练速度上一如既往的快，在几分钟内就可训练完，CNN 则是需要一定时间的，明显慢于 FastText，本文方法由于加入了主题模型，需要提前训练 LDA 模型，速度上是最慢的，时间开销比较高。

5.3.2 issue 分类的有效性分析

为了回答 RQ2，本文需要进行一项人工实验，来验证 issue 分类标签可以帮助开发人员更有效地审阅 issue。具体是选取了 8 位具有 2 年左右开发经验的研究生作为研究人员来参与本次的实验。实验挑选了三个持续关注且熟悉的开源项目，每个项目选择了 10 个 issue，并且将参与者平均分为两组，参与人事先并不知道类别信息。第一组参与人员先不使用标签，只根据标题和描述信息来审阅前 5 个 issue，然后结合生成的分类标签来审阅后 5 个 issue 的内容，第二组则是先使用标签信息来审阅前 5 个 issue，不使用标签来审阅后 5 个 issue，每个参与实验的人需要给出审阅每个 issue 所用的时间。

箱线图 5.5、5.6、5.7 别表示三个项目 issue 的用户研究结果分布，横坐标表示每个项目中选取的 10 个 issue，纵坐标代表审阅 issue 所使用的时间，单位为秒。箱线图可以从五个点来展示出结果的分布情况，并且使用两种颜色将无标签 issue 与有标签 issue 的审阅结果做

了对比。从箱线图中可以看出，使用 issue 分类生成的标签来审阅 issue 使用的时间普遍要比原生 issue 审阅所用的时间少。

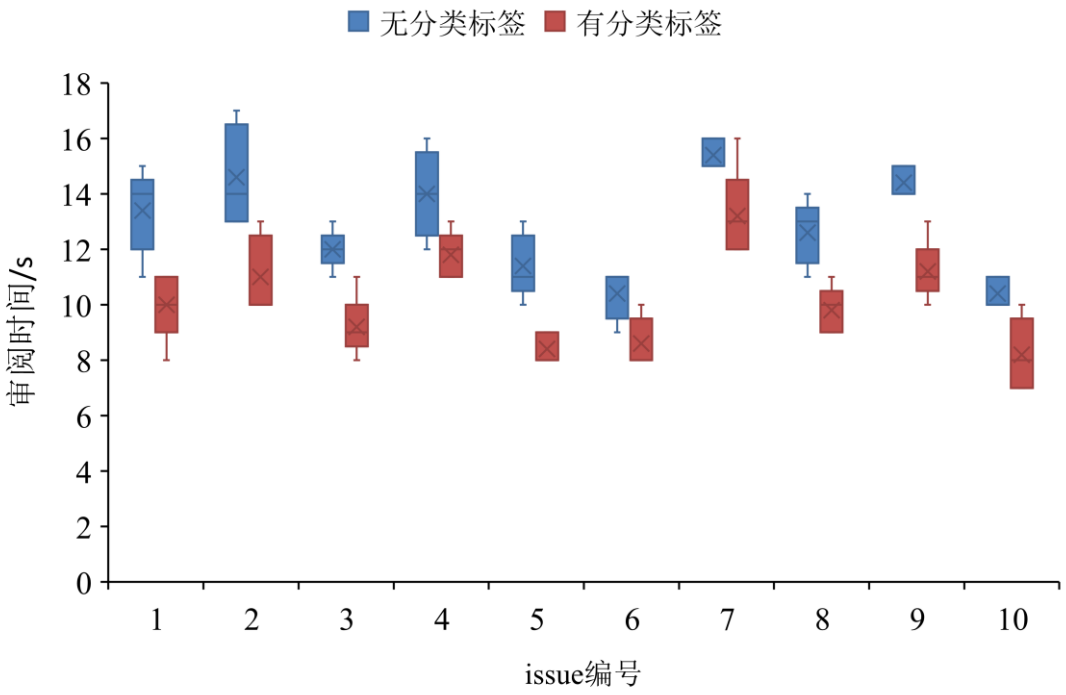

图 5.5 grpc-java 的用户评测结果

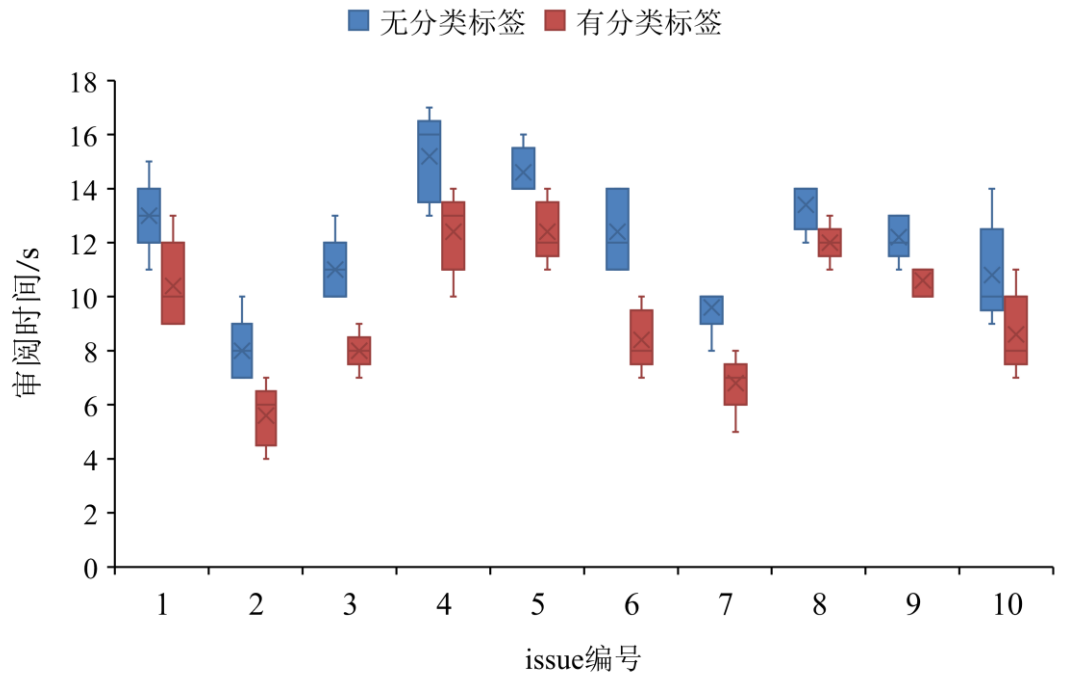

图 5.6 vue 的用户评测结果

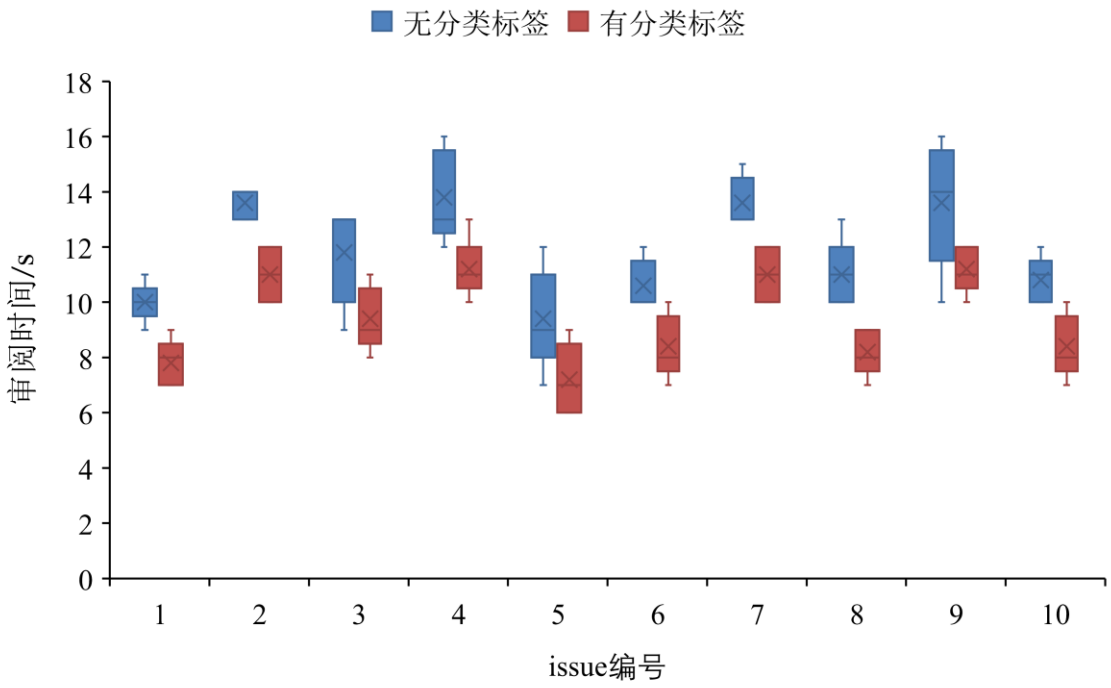

图 5.7 react 的用户评测结果

根据用户评测的结果汇总成表格，如表 5.3 所示，第一列是项目名，第二列是有无分类的标签，no-tag 是没有标签，issue 只有标题和描述信息，tag 是有分类标签；m 代表 issue，数字代表第几个 issue；最后一列 Avg 是审阅每个 issue 的平均时间。从表 5.3 可以看出，在 ant 项目中选取的 issue，在评测人员事先不知道类别的情况下，使用分类标签的平均审阅时间和不使用分类标签的平均审阅时间分别为 12.9 秒和 10.3 秒，缩短了 20%，其他项目中也是如此。总体上来说，带有分类标签信息的 issue 比原生的 issue 更易于审阅。

表 5.3 研究人员审阅 issue 评测表

project	type	m1	m2	m3	m4	m5	m6	m7	m8	m9	m10	Avg
grpc-java	no_tag	14	14.6	12	14	11.4	10.4	15.4	13	14.4	10.4	12.9
	tag	10	11	9	12	8.4	8.6	13	12	11	8	10.3
vue	no_tag	13	8	11	16	14.6	12	9.6	11	12	10	11.7
	tag	10.4	6	8	13	12.4	8	7	8	10.6	8	9.1
react	no_tag	10	13	11.8	13	9	10.6	13.6	11	14	11	11.7
	tag	8	11	9	11	7	8	11	8	11	8	9.2

为了抛开项目的影响因素，本文根据表 5.3，也从纵向分析了 issue 审阅上的差别，如图 5.9 所示，横坐标代表 10 个 issue，纵坐标是审阅 issue 所用使用的平均时间，单位是秒。从图中可以看出，研究人员利用分类标签去审阅 issue 所用的时间都要比不使用分类标签所用的时间要短，表明分类的标签对项目人员审阅 issue 是有帮助的。

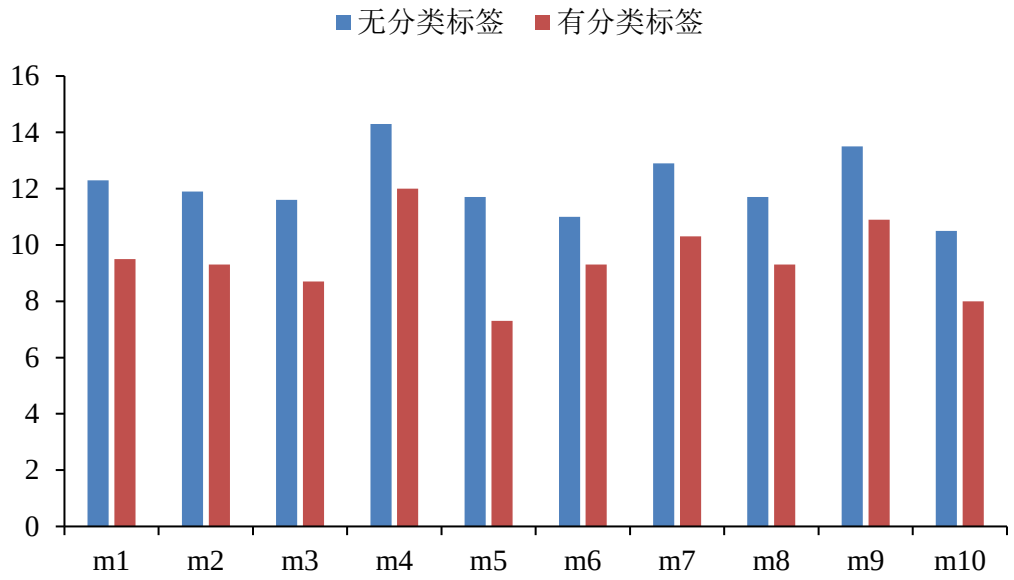

图 5.8 研究人员审阅 10 个 issue 平均使用的时间分布图

为了进一步明确验证有无分类的标签在审阅 issue 上是否存在显著性差异，本文还使用了威尔科克森符号秩检验。Wilcoxon signed-rank test 是一种应用于配对样本的非参数假设检验[52]，用于检验两组数据之间有无差别。本实验将使用分类标签和不使用分类标签审阅 issue 所用的时间进行比较，原假设是两组时间数据之间没有明显差异，备选假设是两组时间数据有明显差异，对其进行假设检验得到 p-value。p-value 代表某一事件发生的概率。给定的显著性水平α，如果 p-value 小于α，则在显著性水平下，拒绝原假设接受备选假设。如果 p-value 大于α，则不拒绝原假设，本实验采用的显著性水平α=0.05。

根据表 5.4 研究人员汇总结果表，从选择的三个项目上看，所有的 p-value 都小于显著水平 0.05，说明两组数据有比较明显的差别，也就是否使用分类的三个标签对于研究者审阅 issue 有明显的区别。

表 5.4 有无分类标签的假设检验

project	p-value
ant	0.0033
commons-lang	0.0032
guava	0.0031

5.4 本章小结

本章主要是对前两章设计的 issue 分类模型进行了实验和结果分析。首先说明了实验环境和评价指标，然后在准确性和有效性上进行验证，从准确性上来讲，本章主要在三个指标上

进行了验证并进行对比分析，从有效性上讲，人工的实验证明了分类的标签在减少审阅 issue 时间开销上的价值。总体而言，工作达到了预期，存在的不足未来会继续完善。

第六章 总结与展望

6.1 本文总结

issue 追踪系统是用户或者外部开发者反馈建议的途径，而配套的标签机制本意是想让提交的 issue 更能直观被了解，但由于标签的分配需要大量依靠人工操作且不同人对标签的分配可能有着不同的想法，标签功能反而让开发团队又爱又恨，使用情况并不理想。本文希望实现一套自动化的 issue 分类系统，完成 issue 在 bug、enhancement、question 上的识别分类，本文主要的贡献如下：

本文提出了基于主题模型和卷积神经网络的 issue 分类方法，首先介绍了 issue 追踪系统方面的研究背景和研究现状，比如 Github 相关研究、issue 分类和 issue 相关研究等。然后介绍了本文实验的背景知识和开发框架，主要包括 issue 追踪系统，主题模型和卷积神经网络。之后介绍了本文数据集的预处理过程，拼接向量的实现方法，依靠 word2vec 和 LDA 分别得到了词向量和 issue 文本的主题向量并实现了融合，构成新的输入特征。

接下来介绍的是 issue 分类的详细实现过程，issue 分类是依托于处理好的数据集，对 issue 实现三个类别的分析，这三个类别是根据已有研究对 issue 标签使用情况的调研建立的。接着介绍了卷积神经网络设计与实现，包括输入层、卷积层、池化层、注意力层和 softmax 分类。

本文的最后进行了实验结果与分析，首先介绍了实验环境、实验过程和评估指标。然后在三个分类指标上对本文模型进行验证，证明准确性的同时进行对比。另外本文还通过人工实验证明了 issue 分类工作的有效性，有着实际的应用价值。

本文工作的意义在于帮助开源平台更有效地管理 issue，促进标签的使用，通过自动化的分配标签取代人工操作，一方面可以让 issue 更有效地被理解，加快处理人员审阅的速度，从而让提出的 issue 及时被解决关闭，避免长期处于 open 状态，另一方面帮助了开发团队节约人力和时间成本，推动软件项目平稳过渡和迭代，帮助代码托管平台繁荣发展。

6.2 展望

本文提出的基于主题模型和卷积神经网络的 issue 分类方法，围绕每个 issue 的标题和描述信息构建数据集，实现了标签的自动分配。相对于现有的 issue 分类技术，本文注意到了每个 issue 文本的主题信息，通过加入主题向量，兼顾了局部和整体语义信息。但是本文所提出

的方法也是具有局限性，所以未来的 issue 自动分类方面的研究工作可以从以下几个方面来进行：

（1）本文使用了卷积神经网络构建分类模型，TextCnn 训练速度快，网络模型结构简单且分类效果不错，未来可以考虑增加模型的深度以进一步提高分类效果同时也可以尝试使用其他的深度学习模型，如 RNN、Bi-LSTM 等。

（2）本文的 issue 分类是单标签的，每条创建的 issue 只会被分类 bug、enhancement 和 question 标签中的一种，但是有些情况下，issue 往往表达了多个的主题，分配多个标签可能更加的合适，所以可以考虑接下来的工作从多标签分类出发。

（3）预训练的语言表征模型 bert 在自然语言处理任务中取得了巨大的成功，刷新了很多的最优记录。issue 的分类任务也可以充分使用 bert 展开，通过 bert 来做 issue 分类，可能能够取得更佳的分类效果。此外，本文的信息主要考虑了标题和描述，issue 发布者的相关信息并没有利用，事实上发布者的履历或者是之前发布的一些 issue 的特征是会一定程度上反应 issue 的主题倾向的，有的人经常发布关于 bug 的 issue，有的人可能更喜欢对代码进行优化等，这些习惯不尽相同，因此今后也可以在实验基础上加入提交者特征来提高分类效果。

参考文献

[1] Pagano D, Bruegge B. User involvement in software evolution practice: a case study[C]//2013 35th International Conference on Software Engineering (ICSE). IEEE, 2013: 953-962.

[2] Panichella S. Summarization techniques for code, change, testing, and user feedback[C]//2018 IEEE Workshop on Validation, Analysis and Evolution of Software Tests (VST). IEEE, 2018: 1-5.

[3] Panichella S, Bavota G, Di Penta M, et al. How developers' collaborations identified from different sources tell us about code changes[C]//2014 IEEE International Conference on Software Maintenance and Evolution. IEEE, 2014: 251-260.

[4] Fan Q, Yu Y, Yin G, et al. Where is the road for issue reports classification based on text mining?[C]//2017 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM). IEEE, 2017: 121-130.

[5] Izquierdo J L C, Cosentino V, Rolandi B, et al. GiLA: GitHub label analyzer[C]//2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER). IEEE, 2015: 479-483.

[6] Bissyandé T F, Lo D, Jiang L, et al. Got issues? who cares about it? a large scale investigation of issue trackers from github[C]//2013 IEEE 24th international symposium on software reliability engineering (ISSRE). IEEE, 2013: 188-197.

[7] Cabot J, Izquierdo J L C, Cosentino V, et al. Exploring the use of labels to categorize issues in open-source software projects[C]//2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER). IEEE, 2015: 550-554.

[8] Guzman E, Azócar D, Li Y. Sentiment analysis of commit comments in GitHub: an empirical study[C]//Proceedings of the 11th working conference on mining software repositories. 2014: 352-355.

[9] Destefanis G, Ortu M, Bowes D, et al. On measuring affects of github issues' commenters[C]//Proceedings of the 3rd International Workshop on Emotion Awareness in Software Engineering. 2018: 14-19.

[10] Zhao G, da Costa D A, Zou Y. Improving the pull requests review process using learning-to-rank algorithms[J]. Empirical Software Engineering, 2019, 24(4): 2140-2170.

[11] Reyes López A.Analyzing GitHub as a Collaborative Software Development Platform: A Systematic Review[J]. 2017.

[12] Gousios G. The GHTorent dataset and tool suite[C]//2013 10th Working Conference on Mining Software Repositories (MSR). IEEE, 2013: 233-236.

[13] Antoniol G, Ayari K, Di Penta M, et al. Is it a bug or an enhancement? A text-based approach to classify change requests[C]//Proceedings of the 2008 conference of the center for advanced studies on collaborative research: meeting of minds. 2008: 304-318.

[14] Herzig K, Just S, Zeller A. It's not a bug, it's a feature: how misclassification impacts bug prediction[C]//2013 35th international conference on software engineering (ICSE). IEEE, 2013: 392-401.

[15] Zhou Y, Tong Y, Gu R, et al. Combining text mining and data mining for bug report classification[J]. Journal of Software: Evolution and Process, 2016, 28(3): 150-176.

[16] Di Sorbo A, Canfora G, Panichella S. " Won't We Fix this Issue?" Qualitative Characterization and Automated Identification of Wontfix Issues on GitHub[J]. arXiv preprint arXiv:1904.02414, 2019.

[17] Alonso-Abad J M, López-Nozal C, Maudes-Raedo J M, et al. Label prediction on issue tracking systems using text mining[J]. Progress in Artificial Intelligence, 2019, 8(3): 325-342.

[18] Fan Q, Yu Y, Yin G, et al. Where is the road for issue reports classification based on text mining?[C]//2017 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM). IEEE,

2017: 121-130.

[19] Kallis R, Di Sorbo A, Canfora G, et al. Ticket tagger: Machine learning driven issue classification[C]//2019 IEEE International Conference on Software Maintenance and Evolution (ICSME). IEEE, 2019: 406-409.

[20] Kikas R, Dumas M, Pfahl D. Using dynamic and contextual features to predict issue lifetime in github projects[C]//2016 IEEE/ACM 13th Working Conference on Mining Software Repositories (MSR). IEEE, 2016: 291-302.

[21] Dhasade A B, Venigalla A S M, Chimalakonda S. Towards prioritizing github issues[C]//Proceedings of the 13th Innovations in Software Engineering Conference on Formerly known as India Software Engineering Conference. 2020: 1-5.

[22] Bühlmann N, Ghafari M. How Do Developers Deal with Security Issue Reports on GitHub?[J]. arXiv preprint arXiv:2112.10359, 2021.

[23] Schreiber A, De Boer C. Modelling knowledge about software processes using provenance graphs and its application to git-based version control systems[C]//Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering Workshops. 2020: 358-359.

[24] Rosenblatt F. The perceptron: a probabilistic model for information storage and organization in the brain[J]. Psychological review, 1958, 65(6): 386.

[25] Kim Y. Convolutional Neural Networks for Sentence Classification[C]. empirical methods in natural language processing, 2014: 1746-1751.

[26] 滕金保, 孔韦韦, 田乔鑫, 等. 基于LSTM-Attention与CNN 混合模型的文本分类方法[J]. 计算机工程与应用, 2021, 57(14): 126-133.

[27] Deerwester S, Dumais S T, Furnas G W, et al. Indexing by latent semantic analysis[J]. Journal of the American society for information science, 1990, 41(6): 391-407.

[28] Hofmann T. Probabilistic latent semantic indexing[C]//Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval. 1999: 50-57.

[29] Blei D M, Ng A Y, Jordan M I. Latent dirichlet allocation[J]. Journal of machine Learning research, 2003, 3(Jan): 993-1022.

[30] 张建华, 梁正友. 基于情感词抽取与LDA特征表示的情感分析方法[J]. 计算机与现代化, 2014 (5): 79-83.

[31] Pedregosa F, Varoquaux G, Gramfort A, et al. Scikit-learn: Machine learning in Python[J]. the Journal of machine Learning research, 2011, 12: 2825-2830.

[32] Gao L, Li C. Hybrid personalized recommended model based on genetic algorithm[C]//In 2008 4th International Conference on Wireless Communications, Networking and Mobile Computing. 2008: 1-4.

[33] Ma L, Zhang Y. Using Word2Vec to process big text data[C]//2015 IEEE International Conference on Big Data (Big Data). IEEE, 2015: 2895-2897.

[34] Muhammad P F, Kusumaningrum R, Wibowo A. Sentiment analysis using Word2vec and long short-term memory (LSTM) for Indonesian hotel reviews[J]. Procedia Computer Science, 2021, 179: 728-735.

[35] Salton G, Wong A, Yang C S. A vector space model for automatic indexing[J]. Communications of the ACM, 1975, 18(11): 613-620.

[36] Wu S. A review on coarse warranty data and analysis[J]. Reliability Engineering & System Safety, 2013, 114: 1-11.

[37] Yogish D, Manjunath T N, Hegadi R S. Review on natural language processing trends and techniques using NLTK[C]//International Conference on Recent Trends in Image Processing and Pattern Recognition. Springer, Singapore, 2018: 589-606.

[38] 吴思竹, 钱庆, 胡铁军, 等. 词形还原方法及实现工具比较分析[J]. 数据分析与知识发现, 2012, 28(3): 27-34.

[39] Vijayarani S, Ilamathi M J, Nithya M. Preprocessing techniques for text mining-an overview[J]. International

Journal of Computer Science & Communication Networks, 2015, 5(1): 7-16.

[40] Haider M M, Hossin M A, Mahi H R, et al. Automatic text summarization using gensim word2vec and k-means clustering algorithm[C]//2020 IEEE Region 10 Symposium (TENSYMP). IEEE, 2020: 283-286.

[41] Do C B, Batzoglou S. What is the expectation maximization algorithm?[J]. Nature biotechnology, 2008, 26(8): 897-899.

[42] Yao K, Zweig G, Hwang M Y, et al. Recurrent neural networks for language understanding[C]//Interspeech. 2013: 2524-2528.

[43] Niu Z, Zhong G, Yu H. A review on the attention mechanism of deep learning[J]. Neurocomputing, 2021, 452: 48-62.

[44] 卢玲, 杨武, 王远伦, 等. 结合注意力机制的长文本分类方法[J]. 计算机应用, 2018, 38(5): 1272-1277.

[45] Park S, Kwak N. Analysis on the dropout effect in convolutional neural networks[C]//Asian conference on computer vision. Springer, Cham, 2016: 189-204.

[46] Salamon J, Bello J P. Deep convolutional neural networks and data augmentation for environmental sound classification[J]. IEEE Signal processing letters, 2017, 24(3): 279-283.

[47] Albawi S, Mohammed T A, Al-Zawi S. Understanding of a convolutional neural network[C]//2017 international conference on engineering and technology (ICET). Ieee, 2017: 1-6.

[48] Eckle K, Schmidt-Hieber J. A comparison of deep networks with ReLU activation function and linear spline-type methods[J]. Neural Networks, 2019, 110: 232-242.

[49] Lai S, Jin L, Yang W. Toward high-performance online HCCR: A CNN approach with DropDistortion, path signature and spatial stochastic max-pooling[J]. Pattern Recognition Letters, 2017, 89: 60-66.

[50] Abadi M, Agarwal A, Barham P, et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems[J]. arXiv preprint arXiv:1603.04467, 2016.

[51] Yao T, Zhai Z, Gao B. Text classification model based on fasttext[C]//2020 IEEE International Conference on Artificial Intelligence and Information Systems (ICAIIS). IEEE, 2020: 154-157.

[52] Woolson R F. Wilcoxon signed-rank test[J]. Wiley encyclopedia of clinical trials, 2007: 1-3.

附录 1 攻读硕士学位期间申请的专利

[1] 张卫丰，徐俊辉.专利名称：基于主题模型和卷积神经网络的 issue 标签分类方法，专利号：202111566439.7, 2021 年 12 月 20 日

致谢

三年的研究生生涯即将结束，心里却是五味杂陈，有对读研生活的不舍，有对未来的无限期盼，也有对导师好友的真诚感谢。回想起看到考研录取名单中有自己名字时欣喜的场面，难以忘怀，选到自己喜欢的导师那更是让我充满斗志。在研究生的最后旅程，回顾过往，有太多值得珍藏的瞬间，有太多值得回味的故事。我要首先感谢南京邮电大学对我的培养，学校的环境和学习氛围令我印象深刻，在这里我逐渐知道该怎么去学习，我适合什么样的工作，对人生有了新的思考。在读软件工程专业过程中，我坚定选择 java 编程的方向，不管是基础知识，如计算机网络、操作系统等的夯实打牢，还是对实际开发使用的框架知识的熟练运用，都对我在秋招中拿到想要的 offer 起到了很大的帮助。

更想感谢的是我的导师张卫丰教授，很感谢老师能够选择我作为自己的学生，我的专业能力并不算优秀，也会犯些小错误，但是张卫丰老师一直包容着我，给我机会，在我最关心的秋招中也给予了我充分的时间去争取工作机会。在论文的完成过程中，老师也对方向进行了细致的把控，围绕着代码仓库的方向，让我顺利的完成课题的研究。一直以来我对老师这一职业充满了敬意，我会永远记得每一个带过我的老师，从小学到现在。

我还要感谢我的同门师兄弟，他们的陪伴让我三年的时光变得很精彩，给予了我极大的帮助。就拿蒋进文师兄，王国程师兄举例，他们很早的就告诉了我应该去学习哪些知识点，如何去准备自己的简历，面试过程中的技巧等，这些都让我不管在找工作或者科研过程中少走了很多弯路。我也要感谢其他实验室的同学，他们都很优秀，有很多值得我学习的地方，能在研究生期间和大家相识相知是我的荣幸。

最后，我要感谢我的家人，他们的支持是我前进的最大动力，他们无私的爱让我倍感温暖，希望以后的自己可以成为家人的依靠，我会一直努力下去的！