中国矿业大学

计算机科学与技术学院

2021 级本科生课程设计报告

课程名称 机器学习应用实践课程设计

设计题目 问题报告分类的多任务学习框架

开课学期 2023-2024 学年第 1 学期

报告时间 2023.11.23

学生姓名 江之昀

学　　号 03210882

班　　级 人工智能 21-1 班

专　　业 人工智能

任课教师 杜文亮

《机器学习应用实践课程设计》课程报告评分表

开课学期:2023-2024 学年第 1 学期

姓名:江之昀　　学号：03210882　　专业班级：人工智能 21-1 班

序号	毕业要求	课程教学目标	考查方式与考查点	占比	得分
1	3.2	**目标 4**：能够根据具体应用，制定机器学习算法设计方案，构建算法训练、验证、测试数据集，并综合考虑机器学习算法模型和软硬件开发环境，进行合理编程实现、程序测试及设计方案优化。	**现场验收与答辩** 考核机器学习算法的分析与选择的合理性、对已选用算法的理解能力，编程实现的代码工作量、设计工作量等。	50%	
2	3.2	**目标 5**：掌握程序设计报告撰写方法，通过成果演示、陈述发言、清晰表达等方式进行有效沟通与交流。	**现场验收与答辩** 考核算法设计成果、所涉及的问题答辩。验收算法设计报告的结构合理性、内容和图表的正确性。验收设计报告排版的规范性。	50%	
总成绩				100%	

任课教师：杜文亮

2023 年 11 月 23 日

问题报告分类的多任务学习框架

江之昀，03210882

（中国矿业大学计算机学院人工智能系，江苏省徐州市 221116）

摘 要 在用户使用一些开源软件项目的过程中，会有产生一些问题（bug）提交给平台请求解决，而平台需要将给问题分配开发者以及问题类型以进行下一步的问题解决。本文旨在解决的是开发者和问题类型分配这两个分类问题。现有的方法独立的处理这两个任务，忽律了其中的相关性。本文复现了一种通过多任务学习（Multi-task Learning, MTL）方法来同时解决两个任务。首先，这两个任务都可以看作是一个基于历史问题报告的分类问题。其次，可以通过多模态，结合问题报告中的 bug 描述文本和代码片段来提高性能。为此，使用文本编码器和抽象语法树（AST）编码器来相应地提取 bug 描述和代码片段的特征。最后，由于训练数据集中类标签的比例不成比例，引入上下文数据增强方法生成语法问题报告，以平衡类标签。本文通过对原始文本数据的数据增强使其在原论文性能的基础上获得了一些进步。

关键词 自然语言处理，多模态，多任务学习，推荐系统

A multi-task Learning Framework for Issue Report Triage

Zhiyun Jiang，03210882

（Department of Artificial Intelligence, School of Computer Science, China University of Mining and Technology, 221116, China）

Abstract In the process of users using some open source software projects, some problems （bugs） will be submitted to the platform for solving, and the platform needs to assign developers and problem types to the problems for further problem solving. This article aims to address the two classification issues of developer and problem type assignment. Existing methods deal with these two tasks independently, and the correlation between them is broken. In this paper, we reproduce a multi-task learning （MTL） method to solve two tasks simultaneously. First, both tasks can be viewed as a classification problem based on historical problem reporting. Second, performance can be improved through multimodality, combined with bug description text and code snippets in problem reports. To do this, a text encoder and an Abstract Syntax Tree （AST） encoder are used to extract bug descriptions and code snippet characteristics accordingly. Finally, due to the disproportionate proportion of class labels in the training data set, a contextual data enhancement method is introduced to generate syntax problem reports to balance class labels. This paper makes some progress on the basis of the performance of the original paper through the data enhancement of the original text data.

Key words Natural Language Processing; Multimodal; Multi-task Learning; Recommendation System

1 引言

软件问题报告，即功能增强请求和在软件维护期间出现的问题通常会存储在 bug 报告库或问题追踪系统中。许多开源软件项目主要使用基于云的问题追踪系统（如 GitHub）来有系统地管理请求。管理问题追踪系统的过程涉及审查新的问题报告,以确保它们有效（从而消除重复报告）,为分配找到合适的开发者,并将每个问题分类到相关的问题类型（例如问题、功能和产品组件）。这个过程也称为问题分类,执行这些任务的人称为分类员或问题跟踪器。实践中,问题跟踪器会重复执行这个过程。问题分类因此很耗时且枯燥,因为许多软件项目由多个开发者维护,并包含各种产品组件。在某些场景下,如果分配的开发者无法解决问题,该问题报告会重新分配给另一个开发者;这个重新分配过程广泛称为问题抛掷（bug tossing）。这个抛掷过程可以增加总体问题修复时间。

1.1 问题阐述

随着每天大量问题报告进入问题追踪系统,手动及时管理这些问题报告变得困难。例如,在 aspnetcore 项目中，在六个月的时间里，报告了 1339 个问题报告,平均每个月 223 个报告。这个项目由 84 位开发者维护,每个报告被归类为 197 个问题类型中的一个，问题跟踪人员需要花费大量时间和精力进行分类。结果，这可能会延迟解决这些问题报告。人们提出了几种自动分类方法,利用候选开发者预测程和问题类型标注过程。

总体来说,现有的问题分类方法主要分为两个类别:代数模型方法和统计语言模型方法。这两种方法都使用单任务学习模型训练开发者和问题类型预测任务。研究中使用词频（TF）和反文档频率（IDF）作为代数模型中的词权重因子。各种距离计算算法（如欧几里得距离）用于计算两个问题报告之间的距离,并通过与现有问题报告匹配将新问题报告与潜在开发者或问题类型建立链接。

这些研究中最常用的代数模型是:向量空间模型（VSM）、laten 语义索引（LSI）和拉丁二项分布（LDA）。更近期的研究探索了统计机器学习表示模型,如支持向量机（SVM）以及神经语言模型,如卷积神经网络（CNN）(Lee 等, 2017) [1] 和循环神经网络（RNNs）,以利用更精确的学习表示。但是,现有方法都将问题报告描和代码片段信息作为连续分布向量表示来捕获,但代码片段的属性没有得到准确描述。此外,这些学习模型的性能可能会由于训练数据的不平衡数据而下降。

1.2 问题背景

本节讨论有关开发者与问题类型推荐任务之间的关系的背景信息，以及它们在问题报告（Issue）和基于拉（Pull Reqeust）的开发项目中的用法。接着，阐述了一些多任务学习的背景。

1.2.1 在问题分类中的开发者和问题类型分配

分配开发者和分配问题类型是错误分类过程中的两个重要任务。在问题跟踪系统中，问题跟踪程序通常执行这两项任务，作为错误分类过程的第一步。

本文的多任务分类模型预测新问题报告的相关开发者和问题类型，用于错误分类过程。问题报告通常包括项目中的错误和需要改进或者增加的功能相关的问题。本文的推荐模型执行如下两项任务。

推荐应分配的开发者任务

此任务涉及预测修复新问题报告的潜在开发者列表。由于问题的复杂性，有时会有多个开发者修复问题报告。

推荐适合的问题类型任务

此任务涉及预测问题类型列表，以便对新问题报告进行分类。例如 GitHub 的问题跟踪系统提供了七个通用标签（即 bug、duplicate、enhancement、help wanted、invalid、question 和 won't fix），同时也可以根据需要添加新的自定义标签。

值得注意的是，大多数项目往往会创建自定义标签来跟踪问题优先级（例如高、低）、产品版本（例如图 1 中的 TF2.14）、状态（awaiting response 等待问题提出者的回复）等标签。

这下标签可以在之后的问题解决中方便提问者和开发者进行检索。

图 1 问题类型标签

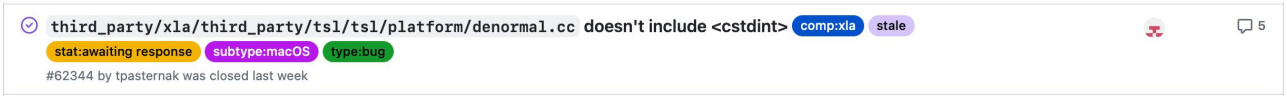


图 2 Issue 报告

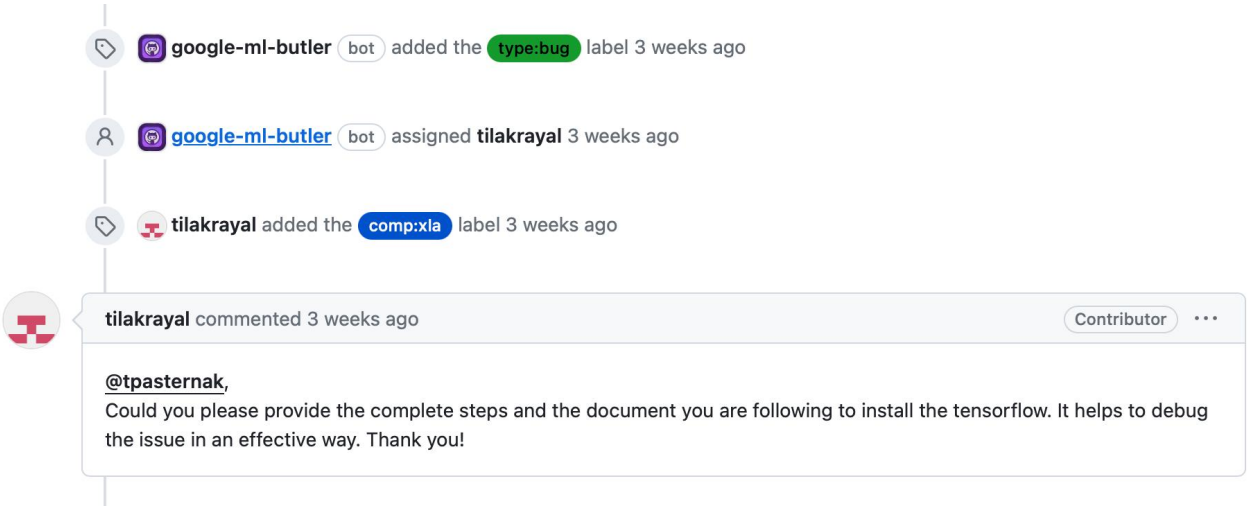


图 3 Issue 处理过程

问题报告处理

图 3 展示了一个 GitHub 问题报告（图 2）的处理过程，可以看到 google-ml-bulter 为问题添加了一个标签 “type: bug”（问题类型分配），分配了一个开发者“tilakrayal”（开发者分配），接下来开发者 tilakrayal 回复了问题提出者。

近年来,开源软件项目中越来越多采用拉取式开发。在拉取式模型中,开发者通过拉取请求表单提交代码以进行代码审查（Code Review）。审核者通常是项目所有者或贡献者,他们作出最后决定是否接受请求中的变更（即拒绝、合并或重新打开）。在 GitHub 项目中,拉取请求表单中的字段与问题请求表单中的字段类似,但也包括额外的部分,如审核者、提交日志选项卡、检查选项卡以及更改文件选项卡。在描述字段中,大多数项目会引用已修复问题 ID 以进行跟踪。审核者字段包含审阅变更的审核者列表,而提交日志选项卡包含提交层次结构,检查选项卡呈现详细的构建输出。最后,更改文件选项卡显示从所有提交更改的文件列表。通过初步观察了解,问题报告中的分配开发者与创建拉取请求来修复问题的开发者可能不同。

本文的目标就是更加准确的推荐可以更好解决问题的开发者，并正确标注问题类型、问题标签。

开发者与问题类型的相关性。

在现有项目中，开发者和问题类型推荐任务都使用历史问题报告来训练预测模型。因此，可以推断出这两个任务之间可以允许存在一个共同的学习表征层，可以一起学习，为使用多任务模型提供了理论基础。

此外，由于软件项目涉及各种组件（例如用户界面、数据库、应用程序接口），问题报告可能涉及系统的任何部分。因此，某些问题类型通常分配给在某些系统领域具有专业知识的一组开发者。如下图 4 展示了一个简单的例子，其中开发者专注于修复特定的系统区域。此例子摘自 aspnetcore。

由这个关系可以得到开发者和问题类型的较强相关性，如果确定一个问题的问题类型，那么就可以去找擅长解决这个问题类型的开发者。同理，如果确定了解决问题的开发者，也可以从这个开发者擅长解决的问题类型中取找这个问题更有可能的问题类型。

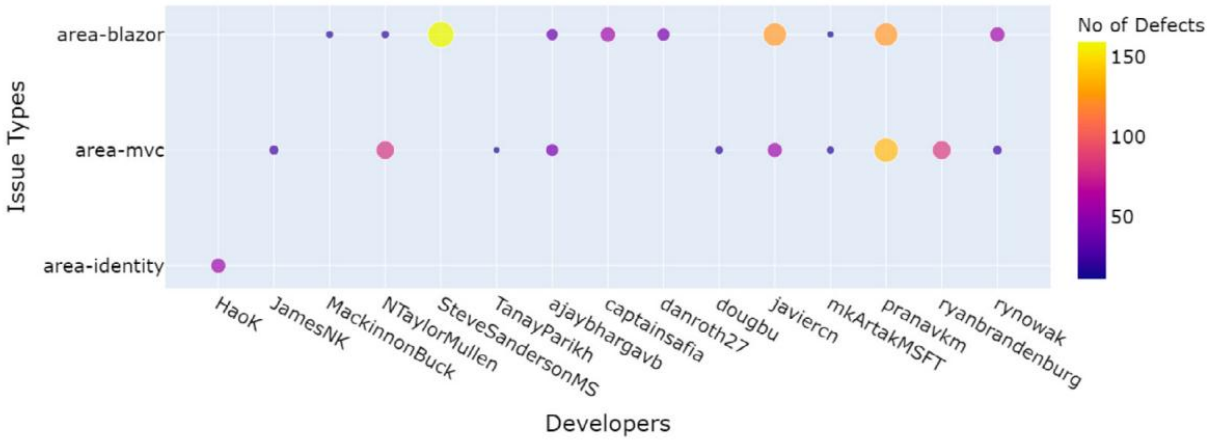


图 4 开发者（Developers）和其解决问题类型（Issue Types）

1.2.1 在问题分类中的开发者和问题类型分配

近年来，多任务学习已成功应用于许多领域，包括计算机视觉（Kokkinos，2017[**2**]；Dvornik 等，2017[**3**]；Bilen 和 Vedaldi，2016[**4**]；Zhou 等，2017[**5**]），自然语言处理（Liu 等，2015[**6**]）和面部识别（Zhang 等，2014[**7**]）。

然而，似乎多任务学习尚未应用于模型化错误分配流程。本文采用多任务学习模型来改善错误分配流程的性能。多任务学习通过共享学习参数，同时解决开发人员和问题类型推荐任务，使这些任务能够相互交互。与独立学习相比，这两个任务的联合学习显著提高了每个任务的性能。

多任务学习模型可以通过硬参数共享或软参数共享隐藏层来共享多个任务之间的参数。硬参数共享模型明确共享所有任务之间的共同学习层，同时将特定于任务的输出层分支化（Caruana，1993[**8**]）。与之相反，软参数共享模型通过规范化每个任务的参数之间的距离来隐式共享这些参数。尽管这两种方法都可以被视为多任务学习模型的基础架构，硬参数共享在神经网络的背景下通常被应用。

本文的多任务学习模型使用了硬参数共享方法，以在共同层学习问题报告的表示，然后分支两个特定任务的输出层以预测开发人员和问题类型。

在共同层，各个问题报告被进一步细分为两类，以有效地学习其特征。

自然语言

问题标题和描述（不包括代码片段）被归类为自然语言，使用上下文编码器来提取特征。

结构语言

代码片段被归为结构语言，使用 AST 编码器提取基本特征。

接下来，将这两个特征结合起来，并输入到任务特定的输出层中执行相应的分类任务。这种方法的详细实现在第 3 节中解释。

2 过往方法

结合分析过去的一些错误报告分类方法，本文旨在解决以下不足:

2.1 数据不平衡带来的偏差（bias）

在真实问题中，往往某些类型的问题和每些开发者解决的问题会占大多数，这会导致数据偏见，让模型更倾向于将新的问题报告分类为那些在以往数据中出现比例更大的类别。

在以往大多数方法中，尝试设置了一个阈值，将低于这个阈值的的类别判定为不经常出现的类别，并将其去除，保持剩下类别的数据均衡。但这样的方法会导致遇到那些不经常出现的类别（即使概率较小）时一定会分类错误。

2.2 对于文本的粗浅认识

在问题描述中有很多类似的表述语态不同（由于数据来源为 Github 等英文平台，会导致时态，单复数等问题）、语序不同，模型会学习到这些特征并给出不同的分类结果。

2.3 过往的工作都忽略了用户提交的问题报告中附带的代码

过往的方法中，大多将问题报告附带的代码片段与问题描述文本一起用同一种方法处理，这种方式忽略了代码片段的结构化特征和描述文本的自然语言阐述这一区别，会导致对代码片段特征学习的欠缺。

2.4 对于开发者分配和错误类型分类这两个具有关联性的任务，浪费时间分别训练两个单独的模型。

推荐解决错误的开发者和确定问题类型是错误分类过程中的两个重要任务。

现有方法分别解决这两个任务,这导致任务重复并忽略任务之间的关联信息。在软件开发项目中,一些开发者通常负责某些组件（如用户界面模块、API 组件）。因此，开发者和问题类型是两个紧密相关的属性。然而，现有的错误分类模型没有充分考
虑这种关联性。

第一，开发者和问题类型标记都可以看作是一个分类问题:它们都依赖于历史错误描述和代码片段信息。

第二，这两个任务的解决都可以为对方提供有利信息：开发者选择可以结合问题类型标记的额外信息来解决问题，确定问题类型可以参考分配的开

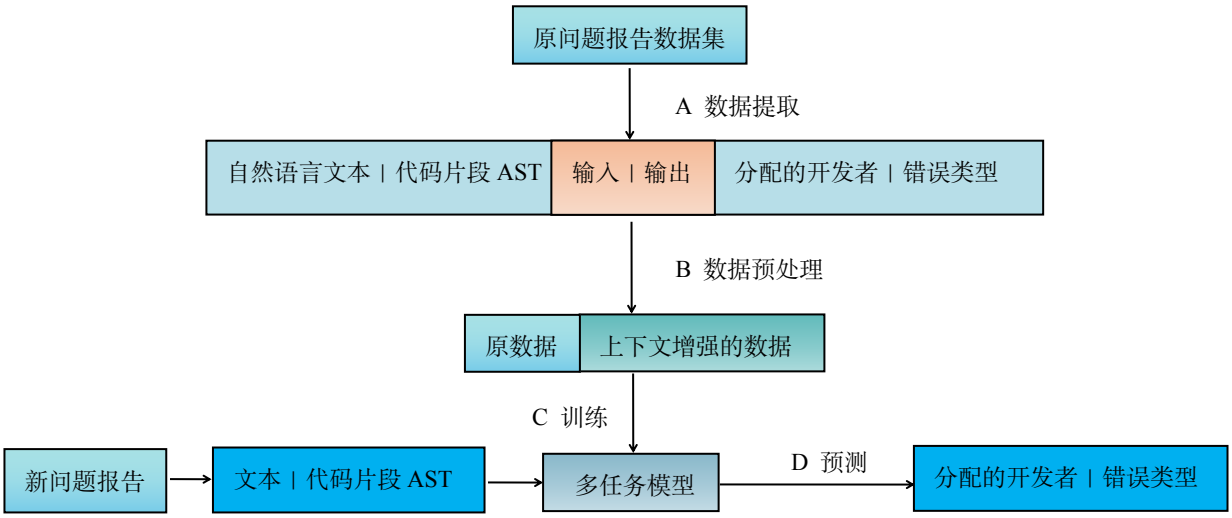


图 5 整体架构

3 本文的方法

本部分先整体介绍如图 5 所示的本文提出方法的整体架构，接着详细对每一部分进行介绍。

训练：

第一步 数据划分：

如图 5 中 A，对于问题报告粒度，提取出其中的文本描述和代码片段和分配开发者、问题类型。对于数据集粒度，划分训练集和测试集。

第二步 数据预处理：

如图 5 中 B，对第一步提取出的自然语言输入和代码片段（AST 形式）输入，开发者输出和问题类型输出，作数据增强，消除类别数量导致的 bias。

接着进行数据清洗，提高性能。

第三步 模型训练：

如图 5 中 C，利用第二步增强后的数据对模型（详见 3.3）进行训练。

预测：如图 5 中 D，对于一份新的报告，将其提取出自然语言输入和代码 AST 输入后即可预测得到

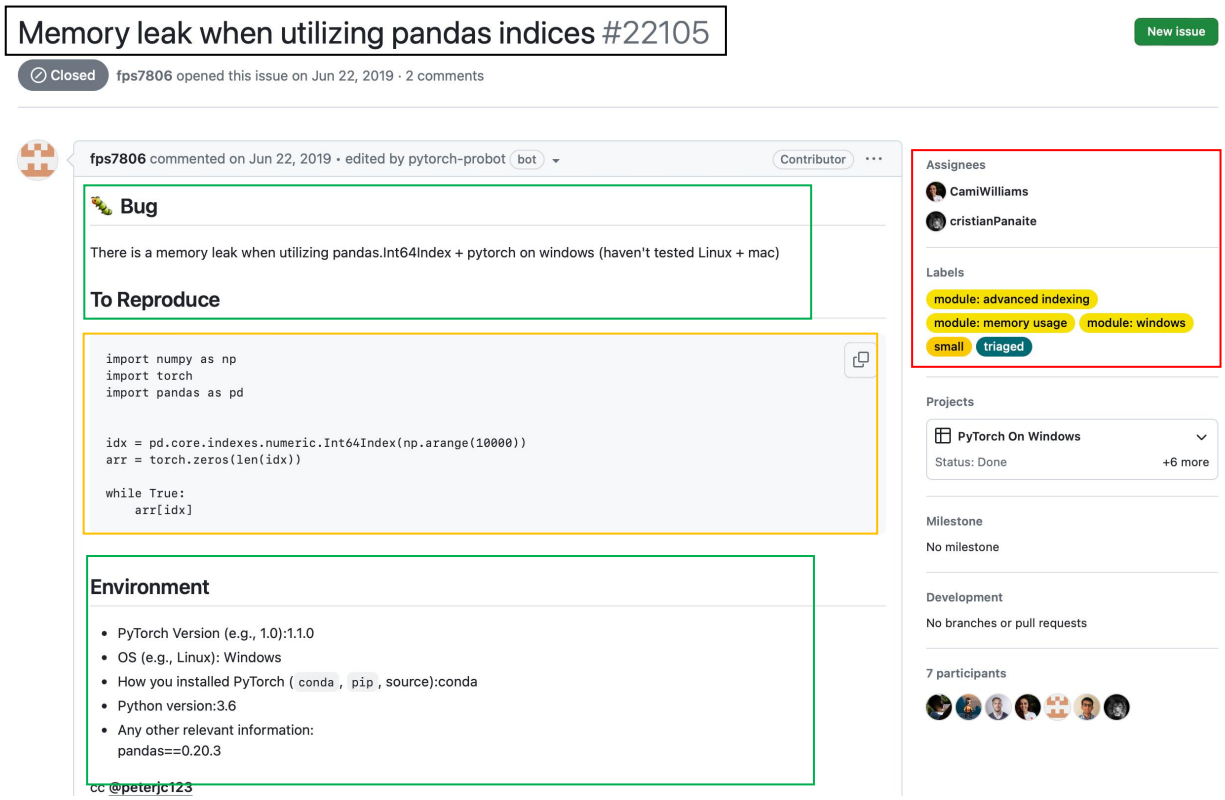


图 6 数据提取样例

应分配的开发者和问题类型。

3.1 数据提取

对于一份如图 6 一样的已经解决（close）的问题报告，对其进行数据提取。

首先可以提取其中解决问题的开发者和问题的标签如图 6 红色标记的。

对于提取自然语言文本，可以在问题报告中找到如图 6 绿色标记的文本，但是注意到问题的标题（图 6 中黑圈）往往是问题的概括，所以将标题和问题内容（自然语言部分）拼接在一起作为了文本部分的输入。

对于代码部分，提取图 6 中的黄色标记部分。但是注意到代码具有结构性，如果直接将不同行连接作为输入会导致结构特征的损失。所以使用 AST（ abstract syntax tree，抽象语法树）进行转换后作为输入。AST 提取器解析每个代码片段并构建 AST 路径。AST 或语法树有两种类型的节点：终端和非终端。终端节点表示用户定义的值（例如标识符），而非终端节点表示语法结构（例如变量声明，for 循环）（Alon 等，2019[**9**]）。AST 路径是终端和非终端节点的序列。在任何问题报告中，由于生成器被修改（Alon 等人，2019），单个代码片段可以包含多个方法，并在每种方法之间添加“〈BM〉”和“〈EM〉”分隔标签以进行模型学习。

3.2 数据预处理

3.2.1 数据增强

在上下文数据增强器中，使用算法 1 中提出的方法为每个项目创建了合成的问题报告。

该算法的输入是问题报告训练集和用于生成合成记录的阈值，输出是包括原记录在内的训练数据集。

在这种方法中，基于训练数据集创建新记录，并且使用阈值参数限制了合成记录的生成。

在这个实验中，使用阈值 30,000 来控制数据增强记录的总数。阈值是根据目标项目的问题报告的大致总数计算出来的。但是，它是一个超参数值，可以根据需要进行更改。

首先，它初始化了大多数和少数类别的值（1 到 3 行）。它通过开发人员和问题类型标签分组来创建聚类。

在初始化之后，将少数类乘以大多数类的样本数来计算估计的合成记录数量，以与阈值数量进行比较（第 4 行）。如果估计值大于阈值，则计算新

的多数类计数值以进行调整（第 5 到 7 行）。

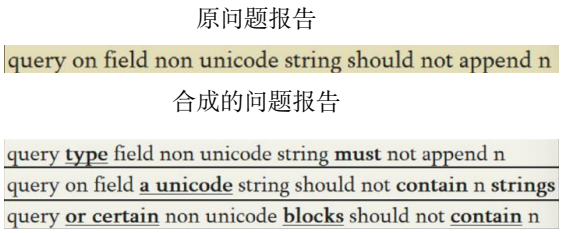


图 7 合成问题报告对比

然后，它迭代遍历每个少数类以生成合成记录（第 8 到 16 行）。在每次迭代中，它随机选择当前少数类的问题报告描述（不包括代码片段）。

然后，它使用 Kafle 等人（2017 年）[**10**]提出的上下文数据增强方法以随机更换描述中的 15%的单词，并创建一个新的问题报告。在这个实验中，使用了已预先训练的 BERT-base-uncased 模型，该模型经过大量英语数据的训练来预测替代词。然而，这种方法也可以推广到其他已预先训练的模型。

最后，该算法的输出是包括原记录和合成记录在内的训练数据集。图 7 通过与原始问题报告进行比较展示了通过上下文数据增强器生成的合成问题报告的示例。在图 7 中，数据增强器生成的所有替换上下文都是用下划线标示的。

总的来说，数据增强器通过替换原始问题报告中的主要关键词来生成合成报告，同时保持原始上下文。

3.2.2 数据清洗

由于在英文平台上获取数据集，在文本中有许多同义词因为大小写，时态，语态而被模型当作不同的特征，导致了不必要的损失，所以对数据进行了词干提取和转换为小写。

在数据集还会出现很多无意义但是出现次数较多的词，比如说 is，the，an 之类的词，仅仅为了让句子语法完整而存在，去除这些词会提高模型的效率和性能。

3.3 多任务模型

如下图 8 为本文的多任务模型示意图。分为三部分：文本编码器（Text Encoder），AST 编码器（AST Path Encoder）和最后的分类器（Classifiers）。

这两个编码器用于基于输入的问题报告生成自然语言和结构（代码）特征。编码器之间的共享层将编码器的输出连接起来，以构建问题报告的整体特征表示。

算法 1. 上下文数据增强算法
输入：原问题报告数据集 Original，阈值 Threshold
输出：用于增强的合成问题报告数据集 New

```
1 MajCCnt ← 样本最多的类别含记录数
2 MinTypeCnt ← 样本少的类别的数量
3 MinCList ← 需要增强的类别
4 DataAugAmount ← 估计要增强的记录数
5 if DataAugAmount < Threshold then
6 |   MajCCount ← (Threshold/DataAugAmount)
  |                *MajCCnt
7 end
8 for minc in MinCList do
9 |   mincCnt←Original 中为 minclass 类的记录数
10|   while mincCnt < Majccnt do
11|   |   sample←随机抽取 Original 中类别为
  |   |          minclass 的一个记录
12|   |   new ← 利用 sample 合成的新记录
13|   |   将 new 加入 New
14|   |   mincCnt ← mincCnt + 1
15|   end
16 end
17 return New
```

最后，分类器分析这些特征，并推荐潜在的开发人员和问题类型作为输出。

模型的主要超参数如下：

batch	32	max_seq_length	300
embedding_dim	100	num_filters	100

批量大小可以设置为从一到几百；这里和 Bengio 在 2012 年的文章[**11**]中一样选择了标准批量大小（32）来训练该模型。

3.3.1 文本编码器 Text Encoder

在该模型中，使用上下文编码器来提取问题报告的自然语言表示。卷积神经网络（CNN）用于生成这些表示。

该编码器的输入是如 3.1 中所述问题标题和描述的拼接在经过 3.2.2 数据清洗后的结果。在输入右侧进行填充，填充到 max_seq_length 范围内来对

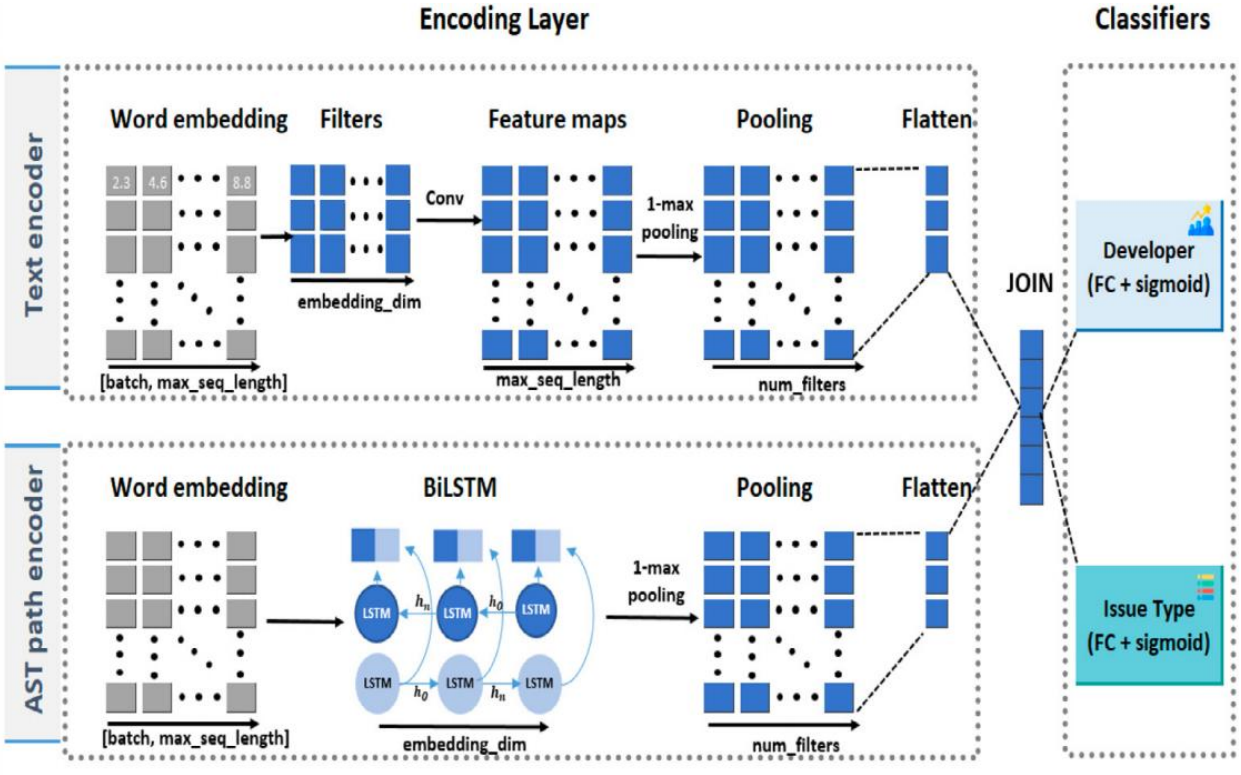


图 8 多分类模型示意图

原始输入进行标准化。

首先，使用 Tonkenzer 将每个问题报告转换成一系列整数（每个整数值是字典中一个标记的索引）将每个问题报告转换为向量。

其次，这些输入被送入具有输入维度的词嵌入层。其中输入维度不作为超参数，而是等于训练集文本转序列的中文本的词典量，也就是输入的文本转来的序列的长度。

接下来的一层是卷积层（filter），使用大小为 2 的卷积核大小来提取重要特征 (Collobert 等在 2011 的文章中提出）。然后，用最大池化层来从每个特征图中提取最相关的信息。

最后，将池化输出传递到连接层进行串联。

3.3.2 AST 编码器 AST Path Encoder

在本模型中，问题报告中的每个代码片段都会被解析，以利用 AST 提取器构建 AST 路径，并将其作为 AST 编码器的输入。

在预处理阶段，AST 路径和文本一样都先转为词向量，具体过程如下。

所有输入都首先在右侧填充到 max_seq_len 长度来准备到相同的大小。其次，AST 路径通过将每个单词转换为整数序列而被转换为向量。然后，这些输入被送入词嵌入层。

为了学习 AST 表示，使用具有长短期记忆（LSTM）神经元的双向递归神经网络（BiLSTM）（Graves 和 Schmidhuber 在 2005 年提出；Cai 等在 2019 年提出）。一般来说，BiLSTM 模型结合两个分开的 LSTM 层，这两个层在相反的方向（即正向和反向）运作，以利用来自先前状态和后续状态的信息。在 LSTM 网络中，每个记忆细胞 c 包含三个门：输入门 i，遗忘门 f 和输出门 o。给定一个输入 AST 序列向量[a1，a2，...，an]，其中 n 表示序列的长度。输入门 i 控制将输入 a[t]保存到当前细胞状态 c[t]。接下来，遗忘门 f 控制将先前细胞状态 c[t−1]中的多少信息保留在当前细胞状态 c[t]中。最后，输出门控制将当前细胞状态 c[t]的多少信息提交给当前输出 h[t]。

接下来，应用最大时间池化操作（Alaeddine 和 Jihene 在 2021 年提出）[**12**]来从 BiLSTM 输出中提取重要信息。最后，输出被展平并送入连接层。在连接层中，两个编码器被串联、输出并送入分类层。

3.3.3 分类器 Classifier

使用 sigmoid 函数对新问题报告进行相关开发人员和问题类型的分类。如图 8 所示，开发人员和问题类型分类器共享相同的结构，但它们的输入标签不同（即开发人员和问题类型）。

因此，在本节中只介绍了一个分类层。分类层由两层组成：一个具有 ReLU 的全连接前馈神经网络 (FFN) 以及一个 sigmoid 层。在 FFN 层中，ReLU 是一种激活函数，如果输入是正的，它会直接输出输入，否则将输出零（Nair 和 Hinton，2010）[**13**]。

接下来，输出向量被送入 sigmoid 层，以预测输入问题报告的适当开发人员或问题类型。接下来，sigmoid 指数激活函数用于计算每个可能类别（即开发人员或问题类型）的概率分布，用于计算 FFN 层输出向量的概率。

4 实验

在本节中，根据第三节中提出的方法编写了代码并设计进行了一系列实验。

4.1 实验准备

首先，在实验开始前，先确定了实验数据集和性能评估指标。

4.1.1 数据集

数据集选择 github 中 aspnetcore 项目的问题报告（Issue Report）。一共含有 7151 份报告，其中 2520 份含有代码片段，一共由 60 个开发者和 131 中类型，时间跨度为 45 天。

在数据选择中，按照先前的研究，只检索具有“已关闭”状态的问题报告（Anvik 等，2006[**14**]；Mani 等，2019[**15**]；Lee 等，2017[**1**]；Xi 等，2018[**16**]），未关闭的问题报告可以视作没有被标注的训练集很难进行充分利用。未分配开发人员或问题类型的问题报告也被删除，因为模型无法训练和验证未标记的记录。此外，排除了分配给“软件机器人”的问题报告，这些报告经常在自动问题分配过程中使用（Golzadeh 等，2020）[**17**]。由于没有实际开发人员使用，这些报告不能用于开发人员预测过程。

4.1.2 性能评估标准

由于本文旨在解决的问题是一个多任务问题（需要同时给出开发者和问题类型），所以设定性能评估标准时要考虑两个输出。

损失 Loss

由于是分类问题，使用 sigmoid 激活函数，二元交叉熵作为损失函数，可以计算出开发者和问题类型的损失。

准确率 Accuracy

分别对开发者和问题类型计算准确率，计算方法如下：

$$\text{Accuracy} = \frac{TN + TP}{TN + TP + FN + FP}$$

精准率 Precision

分别对开发者和问题类型的预测结果计算精准率，公式如下：

$$\text{Precision} = \frac{TP}{TP + FP}$$

召回率 Recall

分别对开发者和问题类型的预测结果计算召回率，公式如下：

$$\text{Recall} = \frac{TP}{TP + FN}$$

汉明损失 Hamming Loss

Hamming Loss 度量的是标签预测和真实标签不相同的比例，即预测错误的标签数占总标签数的比例。因为本文中开发者和问题类型可能由多个，是多标签问题故考虑计算汉明损失。

4.2 消融实验

首先，根据第三节中提出的方法，将其分为数据处理和模型两部分。

4.2.1 数据处理

这里分四点来实现本文提出的方法，本文提出的模型按以下四点来说是 ACUR，设计了 ACU，AUR，ACR，ACU 四种实验和 ACUR 对比，一共五种情况。

数据增强 A

这里用 A（Augment）来简写数据增强，如 3.2.1 中提出的，对数据中那些含记录较少的类型进行上下文替换，增加记录数量以消除不同类型记录数量不一致带来的偏差。

数据清洗 C

这里用 C（Clean）来简写数据清洗，如 3.2.2 中提出的，对数据进行小写化，去除停用词（stopwords），词干提取。

不健全数据 U

这里用 U（Unknown）来简写不健全数据，在

真实情况（例如本文选择的数据集）中，经常会遇到一个问题报告解决了，但是可能由于问题类型难以界定、解决者是匿名的或者是用户，而没有标注问题类型或者解决的开发者。在本文提出的方法中，保存了这些数据，但是设计了当去掉问题类型和开发者没标注的情况的实验，旨在分析模型在面对全部标注的问题时的性能。

移除未知测试集 R

这里用 R（Remove）来简写移除未知测试集的操作，在真实情况（例如本文选择的数据集）中，会遇到测试集中出现训练集没有出现过的标签。类似于现实中遇到一个问题报告，但是它应该被分配的开发者或者问题类型是之前没遇见过的。本文设计实验对比分析了本文模型在这种情况下和去除这些未知测试集情况下的性能。

4.2.2 模型分析

本文提出了如 3.3 节的具有文本编码器，AST 编码器的多任务模型，接下来设计对比实验来分析各组分的作用。

原模型 TCM

这里用 T 来简写文本编码器（Text Encoder），C 来简写代码 AST 编码器（Code Encoder），M 来简写多任务模型。

解藕一个编码器的多任务模型

TM

解耦代码 AST 编码器（Code Encoder）来分析代码 AST 编码器为模型带来的的性能提升。为了避免因为输入信息量减少带来的偏差，将代码片段也输入到文本编码器处理来进行实验。

CM

解耦文本编码器（Text Encoder）来分析文本编码器为模型带来的的性能提升。为了避免因为输入信息量减少带来的偏差，将文本片段也输入到代码编码器处理来进行实验。

4.2.3 基准分析

基准模型 BL

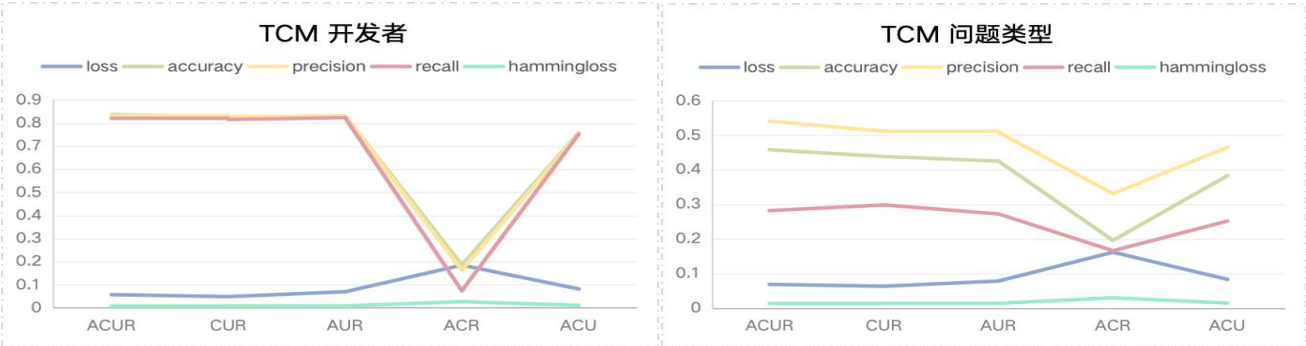
用 BL 来简写基准模型（BaseLine），这里选用在 DeepTriage（Mani 等，2019）[**15**]中提出的，使用递归神经网络（RNN）来学习问题报告的表示，并使用 softmax 层将潜在的开发人员和问题类型推荐为输出的方法作为基准模型进行对比实验分析本文方法的性能提升。

5 实验结果

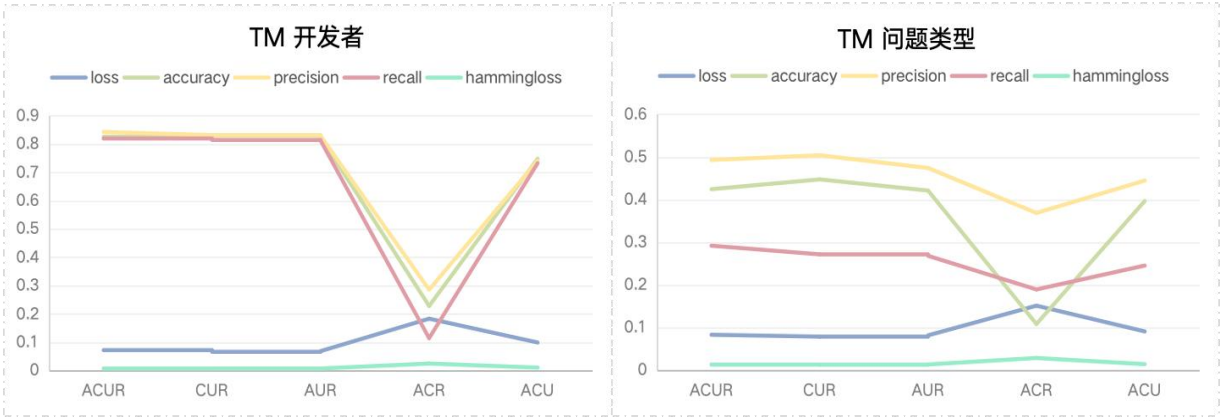
根据第四节中所述的实验设计，对于四个模型：TCM, TM, CM, Baseline 分别对五个数据集 ACUR, CUR，AUR，ACR，ACU 进行三次实验取两个输出的五个性能指标的平均，一共 75 次实验，600 项性能指标数据，结果如下列表所示。

5.1 数据集表现对比

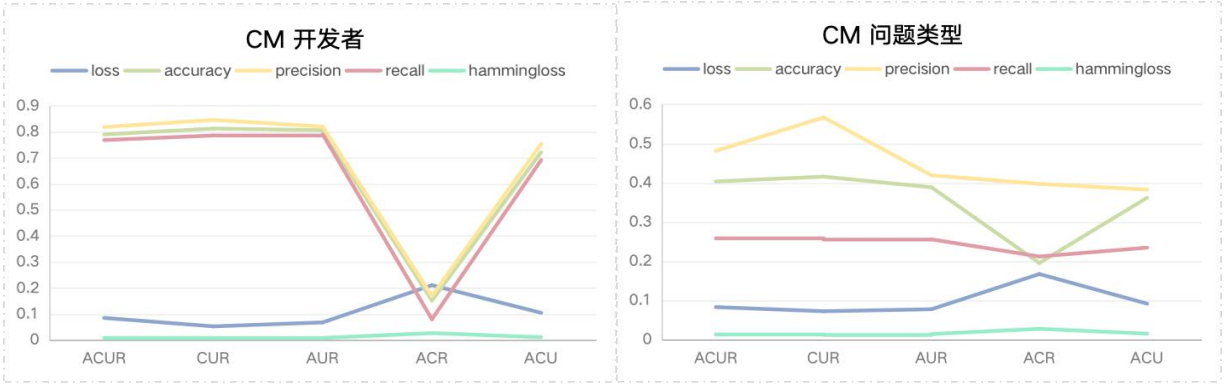
可以看到如图表 1,2,3,4,5 所示，十分明显的是在 ACR 数据集下模型性能的骤降。在 ACR 中，去除了 U 也就是 Unknown 标签的数据。这也导致了不会将结果标记为 Unknown，但是其实这是不符合现实的，因为在面对一个真实输入时，可能结果恰恰是未知的，因为是之前没出现过的问题，这里可以在之后进行更深一步的分析。



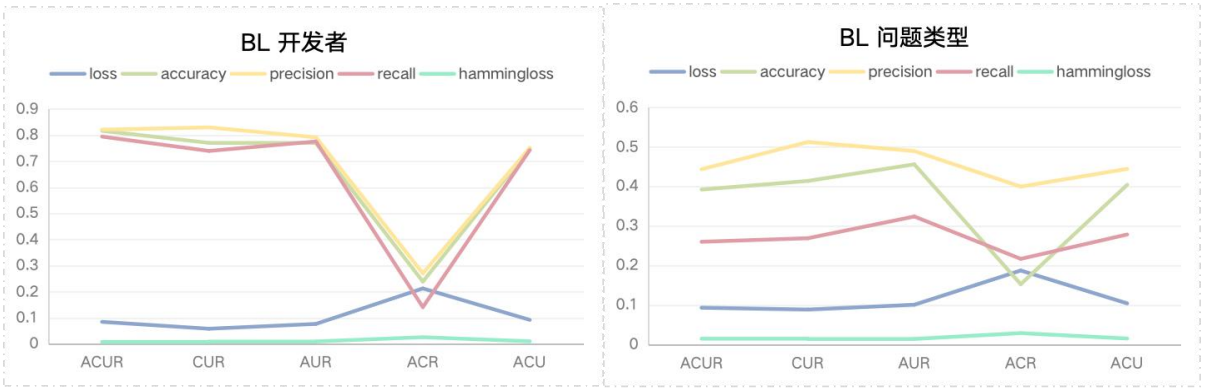
图表 1 TCM 模型在五种数据下对两个问题的性能表现



图表 2 TM 模型在五种数据下对两个问题的性能表现



图表 3 CM 模型在五种数据下对两个问题的性能表现



图表 4 BL 模型在五种数据下对两个问题的性能表现

此外，同一模型下，可以发现 ACUR 的性能较 CUR 和 AUR 有约 2-3 个百分点的提升，较 ACU 有约 8 个百分点的提升。

由此，可以分析得到，数据增强 A 和数据清洗 C 都给模型性能带来了约 2～3 个百分点的提升。

而移除测试集 R 中那些未在训练集中出现的记录给性能带来了约 6～8 个百分点的提升，这时因

5.2 模型表现对比

接着具体如下列表 1，具体观察 Accuracy 指标（分为开发者准确率 Accuracy DEV 和问题类型准确率 Accuracy TYPE）在 75 次实验中的变化，以分析本文方法的性能提升及原因。

Accuracy DEV	ACUR	CUR	AUR	ACR	ACU
TCM	0.8372	0.8238	0.8308	0.1848	0.757
TM	0.8256	0.8298	0.8246	0.2283	0.7483
CM	0.7897	0.8123	0.8058	0.1522	0.722
BL	0.8164	0.7701	0.7692	0.2391	0.75

表 1.1 开发者准确率在不同数据集和模型下的性能表现

Accuracy TYPE	ACUR	CUR	AUR	ACR	ACU
TCM	0.4579	0.4383	0.425	0.1957	0.3829
TM	0.4245	0.4474	0.4212	0.1087	0.3969
CM	0.4034	0.4157	0.3885	0.1957	0.3619
BL	0.392	0.4138	0.4558	0.1522	0.4038

表 1.2 问题类型准确率在不同数据集和模型下的性能表现

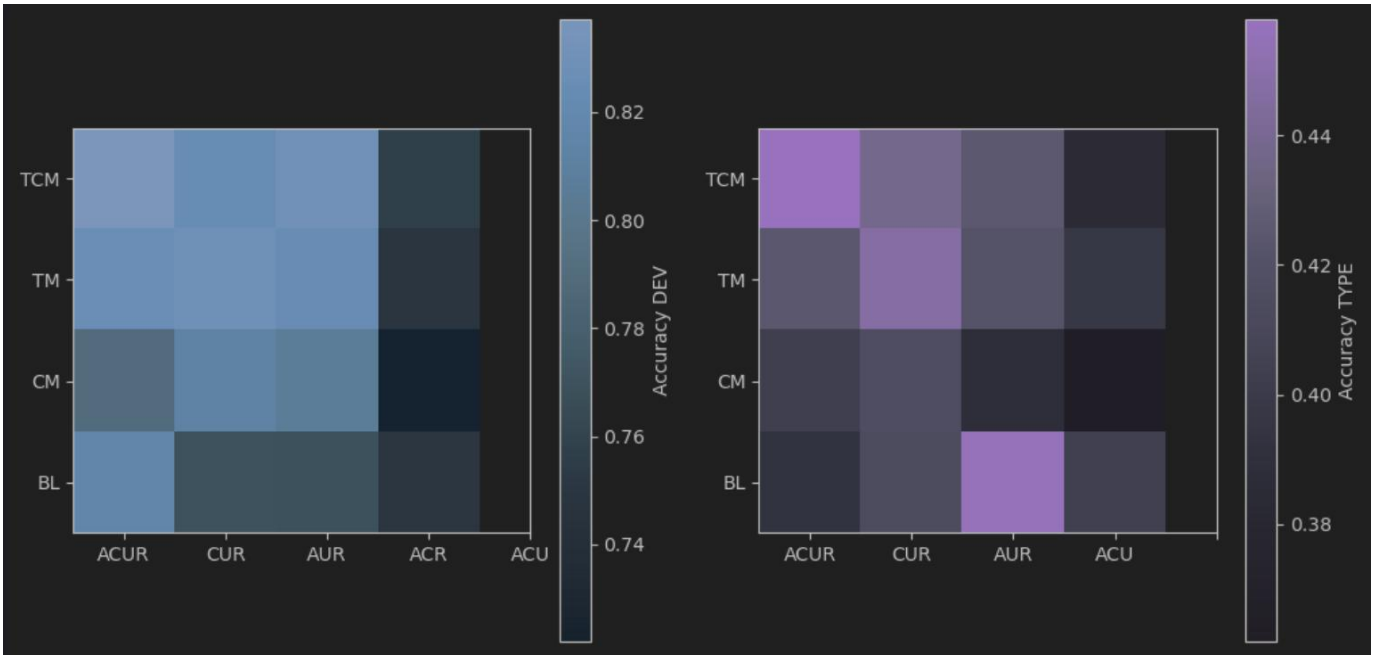


图 9 开发者和问题类型准确率在不同数据集和模型下的性能表现（去除 ACR 数据集）

在 5.1 中已经分析过 ACR 数据集下过低的表现，在图 9 的数据可视化中去掉了 ACR 的数据以增强其余数据的对比性。可以看到 TCM 在 ACUR 下的两项性能是最好的。

对比不同模型，TCM 在四个模型中表现最好，TM 和表现其次，BL 和 CM 的表现比较差。

这首先证明了，含有两个编码器的多任务模型的性能提升。

接着在编码器解藕实验中，文本编码器比代码编码器对模型的性能有较多的提升，但是即使是性能较差的单代码编码器的多任务模型也在两个任务上和基准模型有不相上下的性能。

此外，使用多任务模型还可以对训练时间有提升，在下表 2 和图 10 中展示了这些模型的训练时间，其中 BL DEV 是任务为预测开发者时的基准模型。BL TYPE 是任务为预测问题类型时的基准模型。

TIME	ACUR	CUR	AUR	ACR	ACU
TCM	47	24	47	63	55
TM	16	8	16	24	16
CM	86	39	78	118	78
BL	149	62	142	204	149
BL DEV	78	31	71	102	78
BL TYPE	71	31	71	102	71

表 2 不同数据集和模型训练的时间（s/epoch）

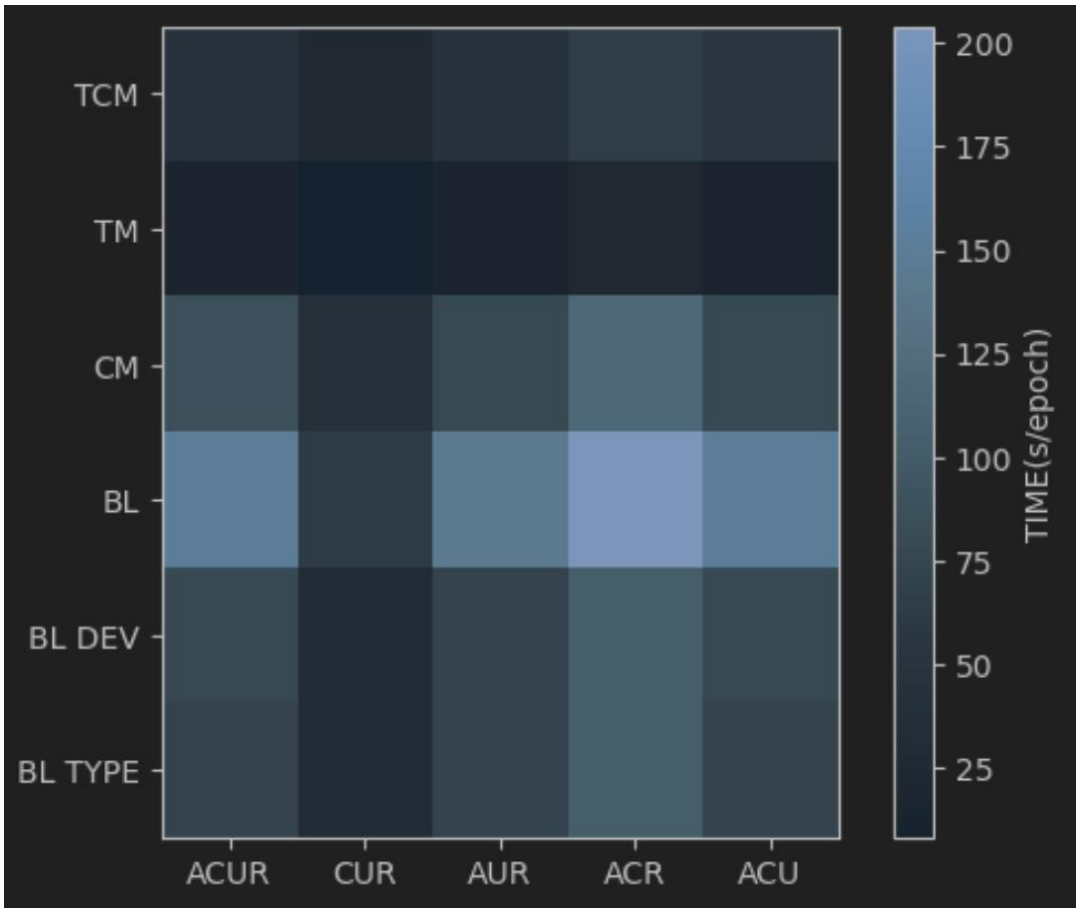


图 10 不同数据集和模型训练的时间（s/epoch）

可以看到 TM 耗时最短，其次是 TCM，而基准模型在执行单任务时的训练时间已经和 TCM 不相上下了，当训练完两个任务的模型后时间已经远远超过 TCM 所需的时间了。

总的来说, TCM 相比 BL 有着约 2 ~ 5 个点的性能提升，同时拥有其两倍的训练速度。

6 总结

在这篇论文中，提出了一个多任务学习框架，用于解决开发者和问题类型分配的分类问题。文章通过多任务学习模型来同时解决开发者分配和问题类型分类两个任务。通过文本编码器和抽象语法树编码器来分别提取 bug 描述和代码片段的特征，并使用上下文数据增强方法来生成语法问题报告，以平衡类标签。在实验中，该框架显示出了较高的性能。文章还分析了不同数据处理和模型解藕对模型性能的影响，以及对模型训练时间的影响。综合实验结果显示，多任务学习模型（TCM）在所设计的实验中性能最佳，并且相比基准模型有显著提升。文章的成果对解决软件问题报告分类问题有实际意义，可以提高问题解决的效率和准确性。

参 考 文 献

[1]Lee, H., Kwon, H., 2017. Going deeper with contehyperspectral image classification. IEEE Trans. Image Process. 26 (10), 4843–4855. http://dx.doi.org/10.1109/TIP.2017.2725580.

[2]Kokkinos, I., 2017. Ubernet: Training a universal convolutional neural network for low-, mid-, and high-level vision using diverse datasets and limited memory. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 6129–6138.

[3]Dvornik, N., Shmelkov, K., Mairal, J., Schmid, C., 2017. Blitznet: A real-time deep network for scene understanding. In: Proceedings of the IEEE International Conference on Computer Vision, pp. 4154–4162.

[4]Bilen, H., Vedaldi, A., 2016. Integrated perception with recurrent multi-task neural networks. In: Advances in Neural Information Processing Systems. pp. 235–243.

[5]Zhou, D., Wang, J., Jiang, B., Guo, H., Li, Y., 2017. Multi-task multi-view learning based on cooperative multi-objective optimization. IEEE Access 6, 19465–19477.

[6]Liu, X., Gao, J., He, X., Deng, L., Duh, K., Wang, Y.-Y., 2015. Representation learning using multi-task deep neural networks for semantic classification and information retrieval.

[7]Zhang, T., Chen, J., Yang, G., Lee, B., Luo, X., 2016. Towards more accurate severity prediction and fixer recommendation of software bugs. J. Syst. Softw. 117, 166–184.

[8]Caruana, R., 1993. Multitask learning: A knowledge-based source of inductive bias. In: Proceedings of the Tenth International Conference on Machine Learning. Morgan Kaufmann, pp. 41–48.

[9]Alon, U., Zilberstein, M., Levy, O., Yahav, E., 2019. Code2vec: Learning distributed representations of code. Proc. ACM Program. Lang. 3 (POPL), http://dx.doi.org/10.1145/3290353.

[10]Kafle, K., Yousefhussien, M., Kanan, C., 2017. Data augmentation for visual question answering. In: Proceedings of the 10th International Conference on Natural Language Generation, pp. 198–202.

[11]Bengio, Y., 2012. Practical recommendations for gradient-based training of deep architectures. Arxiv.

[12]Jiang, G., Wang, W., 2017. Error estimation based on variance analysis of k-fold cross-validation. Pattern Recognit. 69, 94–106.

[13]Nair, V., Hinton, G.E., 2010. Rectified linear units improve restricted Boltzmann machines. In: Proceedings of the 27th International Conference on International Conference on Machine Learning. ICML'10, Omni Press, USA, pp. 807–814.

[14]Anvik, J., Murphy, G.C., 2011. Reducing the effort of bug report triage: Recommenders for development-oriented decisions. ACM Trans. Softw. Eng.Methodol. 20 (3), 1–35.

[15]Mani, S., Sankaran, A., Aralikatte, R., 2019. Deeptriage: Exploring the effectiveness of deep learning for bug triaging. In: Proceedings of the ACM India Joint International Conference on Data Science and Management of Data, pp.171–179.

[16]Xi, S., Yao, Y., Xiao, X., Xu, F., Lu, J., 2018. An effective approach for routing the bug reports to the right fixers. In: Proceedings of the Tenth Asia-Pacific Symposium on Internetware, pp. 1–10.

[17]Golzadeh, M., Legay, D., Decan, A., Mens, T., 2020. Bot or not? Detecting bots in GitHub pull request activity based on comment similarity. In: Proceedings of the IEEE/ACM 42nd International Conference on

Software Engineering Workshops, pp. 31–35.

附录 1. 原始实验数据

TC DEV	ACUR	CUR	AUR	ACR	ACU
loss	0.0567	0.0481	0.0693	0.1843	0.0815
accuracy	0.8372	0.8238	0.8308	0.1848	0.757
precision	0.8321	0.8286	0.8296	0.1631	0.7524
recall	0.8202	0.8153	0.8232	0.0734	0.7506
hammingloss	0.0075	0.008	0.0079	0.0264	0.0102

TC TYPE	loss	accuracy	precision	recall	hammingloss
ACUR	0.0684	0.4579	0.5407	0.2819	0.013
CUR	0.0631	0.4383	0.5113	0.2982	0.0133
AUR	0.0781	0.425	0.5083	0.2722	0.0136
ACR	0.1615	0.1957	0.3309	0.1658	0.0297
ACU	0.083	0.3829	0.4651	0.2516	0.0143

T DEV	loss	accuracy	precision	recall	hammingloss
ACUR	0.0725	0.8256	0.8428	0.8196	0.0075
CUR	0.0666	0.8298	0.8316	0.8144	0.0076
AUR	0.069	0.8246	0.8291	0.812	0.0076
ACR	0.1835	0.2283	0.2861	0.1147	0.0251
ACU	0.0994	0.7483	0.7412	0.7324	0.0108

T TYPE	loss	accuracy	precision	recall	hammingloss
ACUR	0.0835	0.4245	0.493	0.2921	0.0136
CUR	0.0791	0.4474	0.5039	0.2718	0.0134
AUR	0.0821	0.4212	0.4741	0.2684	0.0139
ACR	0.1516	0.1087	0.3688	0.1896	0.0291
ACU	0.0913	0.3969	0.4447	0.2456	0.0146

C DEV	loss	accuracy	precision	recall	hammingloss
ACUR	0.0853	0.7897	0.8181	0.7681	0.0086
CUR	0.0528	0.8123	0.8455	0.7857	0.0081
AUR	0.0681	0.8058	0.8198	0.7901	0.0083
ACR	0.2109	0.1522	0.1694	0.0803	0.0269
ACU	0.1048	0.722	0.753	0.6913	0.0111

C TYPE	loss	accuracy	precision	recall	hammingloss
ACUR	0.0834	0.4034	0.4813	0.2585	0.0137
CUR	0.0729	0.4157	0.5665	0.2554	0.0127
AUR	0.078	0.3885	0.419	0.2559	0.0148
ACR	0.1677	0.1957	0.3972	0.2123	0.0283
ACU	0.0922	0.3619	0.3827	0.2348	0.0157

BL DEV	loss	accuracy	precision	recall	hammingloss
ACUR	0.0848	0.8164	0.8208	0.7942	0.0079
CUR	0.0584	0.7701	0.8294	0.7393	0.0092
AUR	0.0769	0.7692	0.7915	0.7756	0.0097
ACR	0.2128	0.2391	0.2705	0.1412	0.026
ACU	0.0924	0.75	0.7509	0.7425	0.0103

BL TYPE	loss	accuracy	precision	recall	hammingloss
ACUR	0.0927	0.392	0.4434	0.2597	0.0143
CUR	0.088	0.4138	0.5123	0.2687	0.0135
AUR	0.1002	0.4558	0.4894	0.3237	0.0137
ACR	0.1869	0.1522	0.3994	0.2164	0.0284
ACU	0.1037	0.4038	0.4442	0.2781	0.0147

TIME	ACUR	CUR	AUR	ACR	ACU
TCM	47	24	47	63	55
TM	16	8	16	24	16
CM	86	39	78	118	78
BL	149	62	142	204	149
BL DEV	78	31	71	102	78
BL TYPE	71	31	71	102	71

Accuracy DEV	ACUR	CUR	AUR	ACR	ACU
TCM	0.8372	0.8238	0.8308	0.1848	0.757
TM	0.8256	0.8298	0.8246	0.2283	0.7483
CM	0.7897	0.8123	0.8058	0.1522	0.722
BL	0.8164	0.7701	0.7692	0.2391	0.75

Accuracy TYPE	ACUR	CUR	AUR	ACR	ACU
TCM	0.4579	0.4383	0.425	0.1957	0.3829
TM	0.4245	0.4474	0.4212	0.1087	0.3969
CM	0.4034	0.4157	0.3885	0.1957	0.3619
BL	0.392	0.4138	0.4558	0.1522	0.4038