深度学习模型知识图谱构建方法

赵成原[1,2] 刘名威[1,2] 彭 鑫[1,2] 陈碧欢[1,2] 赵文耘[1,2]
[1](复旦大学计算机科学技术学院 上海 200438)
[2](上海市数据科学重点实验室 上海 200438)

摘 要 AI应用开发人员在开发过程中常常需要学习各种深度学习模型，但由于AI领域缺乏学习深度学习模型的统一平台，而且AI应用开发人员并不一定是AI领域的专家，所以从海量的数据中学习深度学习模型是非常不易的。于是提出一种深度学习模型知识图谱构建方法。该方法会抽取模型及实现、组件及实现、组件依赖关系、高层概念、组件类型、组件特性、组件描述、组件开放关系等多种知识，构建模型知识图谱。实验结果表明，知识图谱质量较高，元组正确率达到了90.63%，可以为AI应用开发人员提供一个学习深度学习模型的平台，提高开发效率。

关键词 SE4AI 深度学习模型 组件 知识图谱

中图分类号 TP3 **文献标志码** A DOI: 10.3969/j.issn.1000-386x.2018.01.001

CONSTRUCTION METHOD OF DEEP LEARNING MODEL KNOWLEDGE GRAPH

Zhao Chengyuan[1,2] Liu Mingwei[1,2] Peng Xin[1,2] Chen Bihuan[1,2] Zhao Wenyun[1,2]
[1](*School of Computer Science, Fudan University, Shanghai 200438,China*)
[2](*Shanghai Key Laboratory of Data Science, Fudan University, Shanghai 200438,China*)

Abstract AI application developers often need to learn various deep learning models during the development process, but it is not easy to learn deep learning models from the huge amount of data because the AI field lacks a unified platform for learning deep learning models, and AI application developers are not necessarily experts in the AI field. Therefore, a knowledge graph construction method of deep learning model is proposed. The method extracts a variety of knowledge, such as model and implementation, components and implementation, component dependencies, high-level concepts, component categories, component characteristics, component descriptions, and component open relationships, to construct a model knowledge graph. The experimental results show that the knowledge graph is of high quality with a tuple correct rate of 90.63%, which can provide a platform for AI application developers to learn deep learning models and improve development efficiency.

Keywords SE4AI Deep learning model Component Knowledge Graph

0 引 言

随着人工智能(Artificial Intelligence，简称AI)技术的发展，越来越多的软件应用中都集成了深度学习模型从而实现各种智能化能力，例如智能人机交互[1]、智能推荐[2]、智能决策[3]等。一般而言，AI应用开发人员在开发过程中会去寻找合适的深度学习模型、研究深度学习模型的架构和其中使用的组件以及应用和改造深度学习模型等等。但由于AI领域没有一个完整统一的平台用来学习各种深度学习模型，因此他们常常需要耗费大量精力，从AI论文、开源代码仓库、技术博客中去了解和学习各种深度学习模型。

AI一直是一个非常热门的研究领域，每年都会有许多的研究人员发表大量的AI论文，提出新的AI模型和组件，开源自己的AI模型实现，发布到GitHub代码仓库托管平台。例如，Google研究院将BERT模型的代码实现发布到GitHub仓库中。2019年就有超过12万篇经过同行评审的AI领域的论文发表在各种

收稿日期: 2022-11-09。科技部重点研发项目(2021ZD0112903)。赵成原，硕士生，主研领域：智能化软件开发。刘名威，博士。彭鑫，教授。陈碧欢，副教授。赵文耘，教授。

会议和期刊上，而且数量还在不断增长[4]。2018 年 GitHub 上仓库数量就已经超过了 1 亿。尽管 GitHub 上存在海量的开源代码仓库，但由于很多 AI 应用开发人员本身并不是 AI 技术专家，如何从 GitHub 中筛选出包含 AI 模型的仓库以及识别出其中的模型实现代码，这对于 AI 应用开发人员而言仍然是一个巨大的挑战。

本文提出了一种构建深度学习模型知识图谱的方法，旨在为 AI 应用开发人员提供一个学习深度学习模型的平台，提高 AI 应用开发人员的开发效率。该方法从开源代码仓库中识别 AI 相关的仓库；从仓库代码中抽取模型及其实现；从模型实现代码中抽取组件及其依赖关系；从模型和组件中挖掘高层的概念，并将组件高层概念与组件类型进行链接。另外，该方法还会从组件相关的文本语料中抽取组件的特性、描述和开放关系，最终构建了一个拥有 651,328 个实体节点和 11,810,335 条关系的模型知识图谱。在本文的实验评估中，验证了模型知识图谱具有较高的质量，知识图谱中的元组正确率达到了 90.63%。

1 相关工作

1.1 AI 软件应用问题

随着 AI 技术被集成到各种软件应用中，越来越多的研究者开始关注 AI 应用开发过程中存在的各种软件工程问题。一些研究人员对 Stack Overflow 上 AI 相关的问题帖子进行了经验研究[5]-[6]，发现数据预处理、模型迁移、模型实现、模型部署和程序缺陷等问题是 AI 应用开发中比较常见的问题。Guo 等人[7]分析了由不同架构设计实现的多种深度学习框架和平台，总结了对深度学习软件的开发和部署带来的新挑战。一些研究工作针对 AI 应用开发中出现的程序缺陷进行了详细探究。Humbatova 等人[8]通过分析 GitHub 上的代码提交和流行的深度学习框架以及相关的 Stack Overflow 帖子，对深度学习系统中出现的缺陷进行了分类。Islam 等人[9]对 5 个常见的深度学习框架有关的缺陷修复模式做了深入的分析，并贡献了一个深度学习模型缺陷修复基准数据集。本文的研究工作与他们的都不同，本文主要关注于 AI 应用开发过程中，开发人员需要学习各种深度学习模型，因此提出了构建深度学习模型知识图谱，为 AI 应用开发人员提供一个学习深度学习模型的平台。

1.2 软件工程领域知识图谱

一些软件工程领域的研究人员针对不同的研究目的构建了不同类型的知识图谱。Liu 等人[10]从 API 文档中抽取 API 概念和描述性知识，构造出 API 知识图谱，为特定的自然语言编程任务查询推荐可用的 API 类及方法的文本摘要。Wang 等人[11]提出了一种基于学习的方法，从源代码和软件文档中自动抽取领域词汇表，可以为特定领域构建领域术语知识图谱，为特定领域的知识学习提供帮助。Li 等人[12]从 API 文档中抽取有关 API 使用的警告性知识，并构建了一个 API 警告知识图谱，提高了 API 文档中 API 警告知识的易用性。Liu 等人[13]从 API 文档中抽取与 API 比较有关的知识，构建了 API 比较知识图谱，为可比较的 API 提供比较知识的查询。Xing 等人[14]通过基于 API 文档和软件开发问答文本的概念和关系抽取构造了软件知识图谱，为给定的代码生成语义标签。与这些知识图谱相比，本文提出的深度学习模型知识图谱是一种新型的知识图谱。该知识图谱中包含了模型及实现、组件及实现、高层概念、组件类型等知识，目的是提供给 AI 应用开发人员学习各种深度学习模型。

2 知识图谱构建方法

2.1 知识图谱高层模式

图 1 展示的是模型知识图谱的高层模式。其中，开源仓库节点由 GitHub 开源仓库地址组成；模型实现节点由开源仓库中抽取出的模型实现类组成；模型节点由模型实现类的类名组成；模型高层概念节点由从模型中挖掘出的高层概念组成；组件实例节点由模型实现中具体使用到的某个组件组成；组件节点由模型中使用的组件组成；组件实现节点由抽取出的组件实现类组成；组件使用代码节点由抽取出的组件的使用代码组成；组件高层概念节点由从组件中挖掘出的高层概念组成；组件类型节点由 PapersWithCode 中的组件类型组成；组件特性节点由组件描述性知识抽取方法中抽取出的组件特性组成；组件描述节点由组件描述性知识抽取方法中抽取出的组件描述组成。

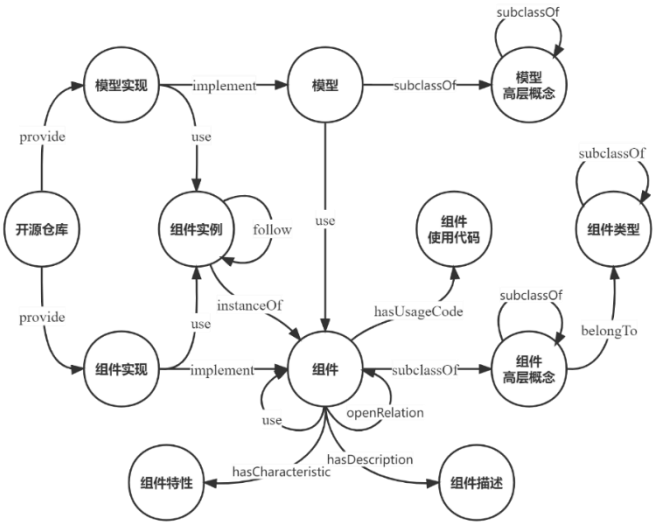

图 1 模型知识图谱高层模式

图 1 中的有向边代表了节点之间的关系。其中，“provide”关系表示开源仓库中提供了模型实现或者组件实现；“implement”关系表示模型的实现代码实现了某一个模型，或者表示某一组件的实现代码实现了某一个组件；“follow”关系表示模型实现中各个组件实例之间存在的依赖关系；“use”关系表示某一模型实现或者某一组件实现中使用了某个组件实例，或者表示某一模型或某一组件中使用了某个组件；“instanceOf”关系表示某一组件和其具体的组件实例之间的关系；“subclassOf”关系表示模型或组件与模型高层概念或组件高层概念之间的类属关系，或者表示组件类型之间的类属关系；“belongTo”关系表示组件高层概念归属于某一种组件类型；“hasUsageCode”关系表示某一个组件拥有的使用代码；“openRelation”关系表示组件之间的开放关系；“hasCharacteristic”关系表示组件和其特性之间的关系；“hasDescription”表示组件和其描述之间的关系。

2.2 方法概述

模型知识图谱构建主要由 AI 仓库识别、模型及实现抽取、组件及依赖关系抽取、高层概念挖掘、组件类型链接、组件描述性知识抽取等多个步骤组成。图 2 展示了整个方法的流程。首先，从开源代码仓库中，利用分类器识别出 AI 相关的开源仓库；接着，从仓库代码中抽取出模型的实现类，再通过抽象语法树分析和启发式规则从模型实现的代码中抽取出组件和组件之间的依赖关系；然后，基于前后缀的模式挖掘方法，从模型和组件中挖掘出高层的语义概念，并将组件的高层概念与 PapersWithCode 中的组件类型进行链接；此外，从组件的文本语料中，利用依存关系解析和词性标注抽取组件的特性，利用启发式规则抽取出组件的描述，基于开放信息抽取(Open Information Extraction ，简称 OpenIE)技术抽取组件之间的开放关系。

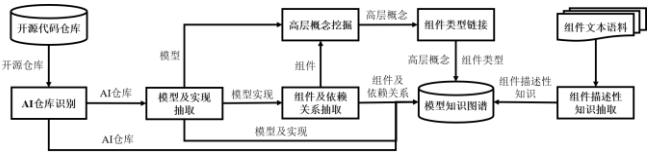

图 2 方法流程

2.3 AI 仓库识别

本文基于人工收集的 ReadMe 数据集训练了一个文本分类器，通过分类开源代码仓库中的 ReadMe 文本，从而识别出 AI 相关的仓库。每个开源代码仓库常常会包含一个 ReadMe 文件。在 ReadMe 文件中，可能记录了开源项目的背景、主要内容、依赖配置、参考文档等诸多重要的信息[15]。这些信息对于了解和使用开源代码仓库非常有用。所以本文通过开源代码仓库中的 ReadMe 文件，来判断该仓库是否是 AI 相关的。为了从开源代码仓库中识别出 AI 相关的仓库，本文利用 PapersWithCode 上的数据，人工构造了 ReadMe 标签数据集，并对比了多种文本分类模型，最终选择使用卷积神经网络(Convolutional Neural Networks，简称 CNN)，从大规模开源仓库中识别出 AI 仓库。

该 CNN 文本分类模型由 Embedding 层、卷积层、池化层、全连接层和 Softmax 层构成。首先，Embedding 层会对输入的开源代码仓库的 ReadMe 文本进行编码，转换成向量表示；接着，卷积层从向量中提取出不同的特征进入池化层；然后，池化层会从提取出的特征中选择出重要的特征，输入全连接层；全连接层将前面提取出的特征综合起来，并输入给 Softmax 层；最后，Softmax 层会输出结果，即输入的 ReadMe 文本是否是 AI 相关的，从而判断对应的代码仓库是否是 AI 仓库。

为了训练 CNN 文本分类模型，本文采用了基于有监督的学习方法。利用 PapersWithCode 提供的仓库链接，爬取代码仓库的 ReadMe 文本，并随机选择了 1,000 个 ReadMe 构造成正样本数据。又从大规模开源仓库中随机选择了 1,000 个 ReadMe 构造成负样本数据，并按照 6:2:2 的比例，将 ReadMe 标签数据集划分成训练集、验证集和测试集。本文基于 Kashgari 框架搭建了一个 CNN 模型，训练了一个对 ReadMe 文本进行二分类的文本分类器。Kashgari 是一个自然语言处理迁移学习框架，常用于文本标记和文本分类。

本文目前只针对 Python 语言的开源仓库进行分析研究，所以预先利用 Scrapy 爬虫框架从 Python 软件库 PyPI 上爬取了 31,520 个 Python 的开源代码仓库。然后利用训练好的 CNN 文本分类模型，通过分类开源代码仓库的 ReadMe 文本，最终抽取出了 4,710 个 AI 仓库。

2.4 模型及实现抽取

本文基于人工构造的数据集训练了一个文本分类器从开源代码中抽取模型实现。模型常常在一个类代码块中进行实现，并且常以模型名作为类名，类中定义前向传播方法，具有较明显的分类特征。考虑到如果采用启发式规则进行抽取会存在一些劣势：一方面，根据模型实现中的分类特征制定启发式规则是复杂且耗时的；另一方面，人工定义的启发式规则难以保证能够全面覆盖所有的特征信息。所以本文采用了基于学习的文本分类方法抽取模型实现。基于学习的文本分类模型不仅可以自动化地学习模型实现中的各种特征，还可以挖掘出一些人工无法发现的隐藏的重要特征。首先，利用启发式规则从开源代码中切分类代码块；然后，利用 PapersWithCode 上的数据，人工构造模型实

现类的标签数据集；接着，基于有监督的学习方法，训练了一个模型实现类的分类器；最后，从大规模的开源代码中抽取模型的实现类。

(1) 类代码块切分

本文主要针对 Python 语言的开源代码，从代码仓库中筛选出 Python 代码文件，使用正则表达式“class \S+.*:(?:(?:\r?\n)+(?:[\t]+\S+.*|[\t]+))+”，按照类的级别对代码进行切割，得到待识别的模型实现类。

(2) 模型实现类标签数据集构造

为了训练模型实现类的分类器，本文基于 PapersWithCode 数据集中的模型和对应的代码仓库，从对应代码仓库中的类代码块中，筛选出类名等于模型名的类代码 18,033 条作为正样本数据。再从其余类代码中人工筛选出同等数量的非模型实现类代码作为负样本数据，并按照 6:2:2 的比例，将类代码标签数据集划分成训练集、验证集和测试集。

(3) 模型实现类分类器训练

本文基于 Kashgari 框架搭建并对比了多种文本分类模型，最终选择使用一个双层的双向长短时记忆网络 (Bi-directional Long Short-Term Memory，简称 BiLSTM)，来识别模型的实现类。图 3 展示了双层的 BiLSTM 结构。该模型由 Embedding 层、双层的 BiLSTM 层、Dropout 层、全连接层和 Softmax 层构成。首先，Embedding 层会对输入的待识别的模型实现类进行编码，转换成向量表示；接着，经过双层的 BiLSTM 层，提取出上下文信息；然后，为了防止过拟合，Dropout 层随机设置某些隐含层节点权重失效，进入全连接层；全连接层将前面提取出的信息综合起来，并输入给 Softmax 层；最后，Softmax 层会输出结果，即输入的类代码是否是模型实现类。

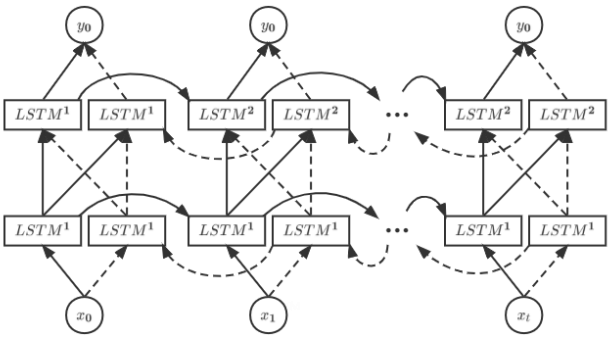

图 3 双层的 BiLSTM 结构

(4) 模型及实现抽取

利用训练好的模型实现类二分类器，最终从开源代码仓库数据集中抽取出了 42,577 个模型实现类，并将该模型实现类的类名作为对应的模型，得到 24,063 个模型。

2.5 组件及依赖关系抽取

本文采用了抽象语法树分析和启发式规则，从模型实现类的代码中抽取出组件实例、组件和组件之间的依赖关系。组件之间常常会进行输入输出传递，在代码中表现为参数之间的传递，可以通过分析代码的抽象语法树抽取组件及其依赖关系。此外，本文还将组件出现的代码作为组件的使用代码，并从模型实现所在的代码仓库中搜索组件的实现代码。

组件抽取

(1) 组件抽取

通过数据观察发现，模型实现中的组件主要声明在模型实现类的构造函数中。所以将模型实现的构造函数转换成抽象语法树，从抽象语法树中“Assign”节点的右子节点“Call”的子树中，抽取出组件实例。接着，对实例的名称按照点号分割，将最右边的简单名作为相应的组件名，并为组件实例和组件之间添加实例关系。图 4 展示了一个模型实现类中构造函数的抽象语法树示例。通过分析抽象语法树中所有的“Assign”节点，从“Assign”的右子节点“Call”的子树中，可以抽取出模型实现类中使用的组件实例有“nn.Dropout”和“LSTM”；接着，对实例名进行处理，得到组件的简单名“Dropout”和“LSTM”作为组件。从“Assign”节点的左子树中，可以抽取出每个组件对应的局部变量分别为“self.dropout”和“self.lstm”，为后面组件关系的抽取提供帮助。最终，共抽取出 222,381 个组件实例节点和 102,969 个组件节点。

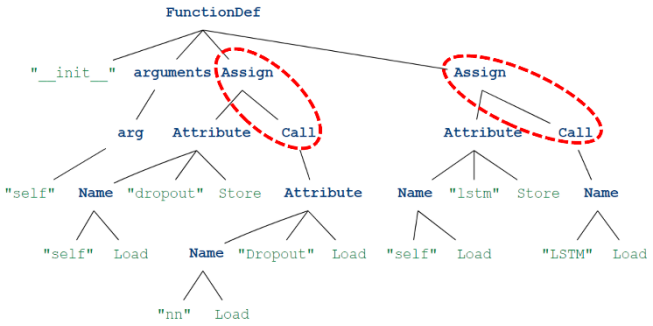

图 4 模型实现中构造函数的抽象语法树示例

(2) 组件依赖关系抽取

利用组件抽取环节得到的组件及其局部变量，并通过分析代码中局部变量之间的参数传递关系，从而抽取出组件之间的依赖关系，例如从“self.dropout(self.lstm(x))”代码中，通过参数分析可以抽取出组件依赖关系“LSTM→nn.Dropout”。最终，共抽取出 118,073 条组件依赖关系。

(3) 组件使用及实现代码抽取

在模型实现类的代码中，抽取组件出现的代码作为组件的使用代码，并在同一个代码仓库中遍历搜索其他的模型实现类代码，如果类名与组件名相同，则将

其作为组件的实现代码。最终，抽取出 29,019 个组件实现代码和 161,970 个组件使用代码。

2.6 高层概念挖掘

为了使模型知识图谱具有更高层的概念理解能力和更深层的语义推理能力，需要从模型和组件中抽取出高层的语义概念。新的 AI 模型和组件常常是建立在旧模型或旧组件的基础之上。它们的名称常常也以旧模型名或旧组件名作为前后缀的形式来命名。所以本文采用了基于前后缀的模式挖掘方法，从抽取出的模型和组件中挖掘出高层的语义概念。

首先，对所有模型或组件节点进行“驼峰式”拆开，得到分词列表；接着，从分词列表中，得到 n-gram 前后缀词集合；然后，将前后缀词集合与模型或组件集合进行匹配，将匹配成功的前后缀词，作为高层概念，添加到高层概念集合中(算法 1. 11~15 行)；最后，为包含高层概念的模型或组件和对应的高层概念之间添加类属关系(算法 1. 16~22 行)，例如经过预处理后的，模型简单名集合[“BERT”，“CodeBERT”，“SciBERT”，“VideoBERT”]，再经过“驼峰式”拆开和 n-gram 处理后得到前后缀词集合[“BERT”，“Code”，“Sci”，“Video”]，其中与模型集合匹配成功的词为“BERT”，所以将“BERT”作为高层概念，并且将“CodeBERT”，“SciBERT”和“VideoBERT”名称中包含了“BERT”的模型，作为“BERT”的一个子类型，并添加“subclass of”类属关系。最终，共挖掘出 20,189 个模型概念和 19,880 个组件高层概念。

算法 1 高层概念挖掘算法。

输入：模型或组件集合 *ComponentSet*
输出：高层概念关系图 *G*

```
1：function ConceptMining(ComponentSet)
2：    prefixSuffixWordSet //模型或组件的 n-gram 前后缀词
3：    ConceptSet //挖掘出的高层概念
4：    for component in ComponentSet:
5：        tokens ← camelSplit(component)
6：        words ← getNgramPrefixSuffixWords(tokens)
7：        for word in words:
8：            prefixSuffixWordSet.add(word)
9：        end for
10：    end for
11：    for word in prefixSuffixWordSet:
12：        if word in ComponentSet:
13：            ConceptSet.add(word)
14：        end if
15：    end for
16：    for concept in ConceptSet:
17：        for component in ComponentSet:
18：            if concept in component:
19：                向 G 中添加模型或组件和高层概念间的关系
20：            end if
21：        end for
22：    end for
23：return G
```

2.7 组件类型链接

图 5 展示的是 PapersWithCode 中的组件类型信息，其中包含了组件类型、组件类型描述、所属组件等信息。将知识图谱中的组件高层概念关联到 PapersWithCode 中的组件类型，也可以增强知识图谱的概念理解和语义推理能力。所以本文将抽取出的组件高层概念和 PapersWithCode 中的组件类型进行链接。

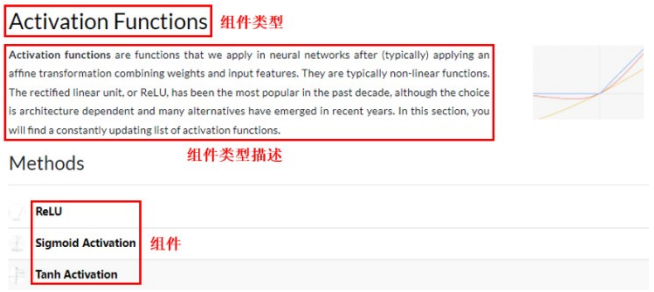

图 5 PapersWithCode 组件类型信息示例

算法 2 详细描述了组件高层概念与 PapersWithCode 组件类型之间的链接。该算法主要通过衡量模型知识图谱中的组件高层概念与 PapersWithCode 组件类型之间的相似性来进行链接。主要步骤如下：首先，经过预处理，从 PapersWithCode 提供的组件数据集中，得到了 350 个组件类型以及 1,802 个组件，并将其转换成所属组件-组件类型映射作为算法的输入(算法 2. 2 行)；然后，依次遍历组件高层概念和组件类型信息中的每个组件，进行关键词匹配，如果匹配成功，则直接将高层概念链接到对应的组件类型，否则利用 Yamada 等人[16]预训练的 Wikipedia 词向量对高层概念和组件进行向量化表示，同时利用文本相似度工具库 TextDistance 提供的基于 token 的 Jaccard 算法，通过计算向量之间的余弦相似度和 Jaccard 文本相似度来衡量高层概念和组件之间的相似性。计算方法分别见公式(1)、(2)。若两种相似度之和大于自定义的相似度阈值，则将高层概念链接到对应的组件类型，返回链接后的组件高层概念-组件类型映射(算法 2. 6~16 行)。最终完成了 3,838 条链接关系。

定义$Sim\ cos(C_1, C_2)$为知识图谱中组件高层概念C_1和 PapersWithCode 中组件C_2的余弦相似度，其计算公式为：

$$Sim\cos(C_1, C_2) = \frac{V(C_1) * V(C_2)}{\|V(C_1)\|^2 * \|V(C_2)\|^2} \quad (1)$$

其中，$V(C_1)$和$V(C_2)$分别为组件C_1和组件C_2的词向量。

定义$Sim\ jac(C_1, C_2)$为知识图谱中组件高层概念C_1和 PapersWithCode 中组件C_2的 Jaccard 文本相似度，其计算公式为：

$$Sim\ jac(C_1, C_2) = \frac{|C_1 \cap C_2|}{|C_1 \cup C_2|} \tag{2}$$

算法 2 高层概念与组件类型链接算法。

输入：组件集合 *ComponentSet* 和 PapersWithCode 所属组件-组件类型映射 *CategoryMap*

输出：完成链接后的高层概念-组件类型映射 *CompleteMap*

```
1：function Linking(ComponentSet, CategoryInfo)
2：    获取 PapersWithCode 中的组件集合 PWCComponentSet
3：    semanticSim //记录组件之间的语义相似度
4：    textSim //记录组件之间的文本相似度
5：    Threshold //自定义的相似度阈值
6：    for comp in ComponentSet:
7：        for pwcComp in PWCComponentSet:
8：            semanticSim ← getSemanticSim (comp, pwcComp)
9：            textSim ← getTextSim (comp, pwcComp)
10：           if semanticSim + textSim > Threshold:
11：               category ←CategoryMap.get(pwcComp)
12：               CompleteMap.put(comp, category)
13：           end if
14：       end for
15：   end for
16：return CompleteMap
```

2.8 组件描述性知识抽取

为了丰富模型知识图谱，同时为了增强模型参考实现的可解释性。本文从 PapersWithCode、Wikipedia、Wikidata 和 AI 相关的论文摘要中，筛选出组件相关的文本语料。Wikipedia 即维基百科，它是一个可自由编辑的，用多种语言编写而成的网络百科全书。Wikidata 是一个自由的协作式多语言辅助知识库，旨在为维基百科、维基共享资源以及其他的维基媒体项目提供支持。基于依存关系解析和词性标注抽取组件的特性，基于抽取出的特性和启发式规则抽取组件的描述，基于 OpenIE 方法抽取组件之间的开放关系，从而为组件添加更多的描述性知识。首先，需要对组件文本语料利用自然语言处理工具 NLTK 进行分句，并筛选出包含组件名的句子作为后续抽取方法的输入。

(1) 组件特性抽取

本文利用自然语言处理工具 spaCy 对组件语料中的句子进行依存关系解析和词性标注。首先，解析出句子中的主语，如果主语为组件，则从该句子中抽取出形容词或副词或者名词短语作为组件的特性，例如组件句子“SGD is an optimization technique and is fast.”的依存关系解析和词性标注如图 6 所示。其中，主语为组件 SGD，抽取名词短语“optimization technique”和形容词“fast”作为组件 SGD 的特性。最终，共抽取出 4,087 个组件特性节点。

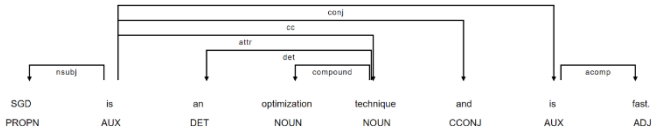

图 6 依存关系解析和词性标注示例

(2) 组件描述抽取

利用组件特性抽取步骤中，抽出的组件特性，并基于启发式规则，从组件语料中，抽取以组件名开头，或者同时包含组件名和组件特性的句子作为组件的描述，例如组件句子“SGD is an optimization technique and is fast.”是以组件名“SGD”开头的，而且也包含了组件的特性，所以该句将作为组件 SGD 的一个描述性句子。最终共抽取出 2,484 个描述性句子作为组件描述节点。

(3) 组件开放关系抽取

利用 Stanford NLP 的 OpenIE 工具从组件语料中抽取带有得分的(头实体，关系，尾实体)三元组开放关系，例如抽取出的得分为 1.0 的三元组开放关系(LSTM, stronger than, GRU)，并筛选出得分高的组件实体之间的开放关系，来补充知识图谱中组件之间的关系。最终，共抽取出了 1,211,914 条三元组开放关系，并从中筛选出头实体和尾实体都是组件且得分大于 0.8 的共 163,164 条开放关系，作为组件之间的开放关系。

2.9 知识图谱构建结果

通过以上的知识图谱构建步骤，本文生成了一个拥有 651,328 个实体节点，1,810,335 条关系的模型知识图谱。其中，实体节点包括了 21,359 个开源仓库节点，42,577 个模型实现节点，29,019 个组件实现节点，24,063 个模型节点，20,189 个模型高层概念节点，102,969 个组件节点，222,381 个组件实例节点，198,80 个组件高层概念节点，161,970 个组件使用代码节点，350 个组件类型节点，4,087 个组件特性节点和 2,484 个组件描述节点。关系包括了 119,654 条 subclass of 关系，3838 条 belong to 关系，664423 条 use 关系，121363 条 provide 关系，126463 条 implement 关系，222381 条 instance of 关系，270976 条 has usage code 关系，118073 条 follow 关系和 163,164 条开放关系。

3 实 验

为了评估模型知识图谱的质量， 本实验对知识图谱构建中的两个重要步骤 AI 仓库识别和模型实现类

抽取的效果进行评估。此外，还对知识图谱中元组准确性进行评估。

3.1 实验设计

本文通过对比不同的深度学习模型在 AI 仓库识别和模型实现类抽取上的效果，以及模型知识图谱中元组的正确性来评估该知识图谱的内部质量。元组表示了两个实体之间的关系，例如元组 (LSTM, belongTo, Recurrent Neural Network)表示实体 LSTM 是属于实体 Recurrent Neural Network。

(1) AI 仓库识别实验设计

本文基于 Kashgari 框架实现了 5 种常见的文本分类模型，并与本文中采用的 CNN 模型进行对比。将每种模型在相同的 ReadMe 文本数据集上进行训练测试。在实验中，每种模型都采用了相同的超参数，例如批处理大小为 128、最大迭代次数为 30 次等。此外，为了防止过拟合，本文采用了早停法进行训练，验证了不同模型在 AI 仓库识别上的效果。5 种对比模型如下：

1) CNN+GRU：将循环神经网络与门控循环单元 (Gated Recurrent Unit，简称 GRU) 进行组合；

2) CNN+LSTM：将循环神经网络与长短时记忆网络 (Long Short-Term Memory，简称 LSTM) 组合；

3) BiGRU：双向门控循环单元 (Bi-directional Gated Recurrent Unit，简称 BiGRU)；

4) BiLSTM：双向长短时记忆网络；

5) 双层 BiLSTM：在第 4 个对比模型基础上，再增加一层 BiLSTM，实现双层的 BiLSTM 模型。

(2) 模型实现类抽取实验设计

本文将基于 Kashgari 框架实现的 CNN、CNN+GRU、CNN+LSTM、BiGRU 和 BiLSTM 共 5 种分类模型，与本文中采用的双层 BiLSTM 模型对比。在相同的模型实现类数据集上，采用相同的超参数进行训练，验证不同模型在模型实现类抽取上的效果。

(3) 知识图谱元组质量实验设计

与以前的研究[11]-[13]类似，本文采用了一种随机抽样的方式以确保在样本中观察到的比例，能够在一定的置信水平和置信区间下反映出总体的表现。在 95% 的置信水平和置信区间为 5 的条件下，所需的样本量为 384 个。所以本文随机地从模型知识图谱中抽取了 384 个关系元组，并确保每种关系被抽取的元组数量尽可能均匀。然后，邀请两名熟悉 AI 应用开发，但没有参与本文研究工作的研究生分别独立标注采样的元组。对于每个元组，他们需要对每个元组结果的正确性进行标注。如果他们的标注结果不同，则让第三名熟悉 AI 应用开发的研究生提供仲裁，以解决标注结果冲突。

为了验证两名研究生人工标注结果的一致性，本文还计算了两人标注结果的 Cohen's Kappa[17]系数。通常 Kappa 值大于 0.6 即满足一致性要求。

3.2 实验结果

(1) AI 仓库识别实验结果

在 AI 仓库识别实验中，本文采用的 CNN 模型与其他模型相比，在精确率、召回率、F1 值和准确率上都表现更好，达到了 98.94%。表 1 中展示了具体的对比结果。少量识别错误的仓库并不一定能够在后面的模型实现类抽取步骤抽取出模型实现类，所以对整体的结果影响较小，在可接受范围之内。

表 1 不同模型的 AI 仓库识别结果对比

模型	精确率 (%)	召回率 (%)	F1 值 (%)	准确率 (%)
CNN	**98.94%**	**98.94%**	**98.94%**	**98.94%**
CNN + GRU	76.18%	60.76%	67.60%	60.76%
CNN + LSTM	75.49%	55.76%	64.14%	55.76%
BiGRU	98.33%	98.33%	98.33%	98.33%
BiLSTM	98.06%	98.03%	98.04%	98.03%
双层 BiLSTM	97.32%	97.27%	97.29%	97.27%

(2) 模型实现类抽取实验结果

在模型实现类抽取实验中，本文采用的双层 BiLSTM 模型与其他深度学习模型相比，在精确率、召回率、F1 值和准确率上都表现更好，达到了 98.00%。表 2 中展示了具体的对比结果。少量识别错误的模型实现类也并不一定能够在后面的组件抽取步骤抽取出组件，所以对整体的结果影响也较小，也在可接受的范围之内。

表 2 不同模型的模型实现类抽取结果对比

模型	精确率 (%)	召回率 (%)	F1 值 (%)	准确率 (%)
CNN	97.69%	97.69%	97.69%	97.69%
CNN + GRU	94.87%	94.57%	94.72%	94.57%
CNN + LSTM	97.70%	97.70%	97.70%	97.70%
BiGRU	97.30%	97.30%	97.30%	97.30%
BiLSTM	97.85%	97.85%	97.85%	97.85%
双层 BiLSTM	**98.00%**	**98.00%**	**98.00%**	**98.00%**

(3) 知识图谱元组质量实验结果

两名研究生标注者的 Cohen's Kappa 一致性为 0.78，表明标注结果的一致性较好。在最终的标注结果中，有 348 个元组被标注的是正确的，36 个被标注是不正确的，元组的正确率为 90.63%。本文分析了这些被标注为错误的元组数据，发现这些错误的原因存在于模型知识图谱构建的不同步骤中，例如在 GitHub 仓库中错误地抽取出了模型实现，这是由模型实现类的分类器

在可接受范围内的误判造成的；还有一些组件的高层概念与组件类型链接出现了错误，这与本文定义的相似度阈值的大小有关。从整体上来看，知识图谱中的元组是高质量的。

3.3 实验总结

模型知识图谱的内部质量很高，AI 仓库识别的准确率达到了 98.94%，模型实现类抽取的准确率达到了 98.00%，抽样的 384 个元组的正确率达到了 90.63%，知识图谱构建的每个步骤总体上是非常有效的。

4 结 语

本文提出了一种深度学习模型知识图谱构建方法。该方法通过 AI 仓库识别、模型及实现抽取、组件及依赖关系抽取、高层概念挖掘、组件类型链接、组件描述性知识抽取等步骤构建了一个包含开源仓库、模型及实现、组件及实现、组件关系、组件类型、高层概念等多种知识的模型知识图谱。实验结果表明，该模型知识图谱质量很高，知识图谱中的元组正确率达到了 90.63%。模型知识图谱为 AI 应用开发人员提供了一个学习深度学习模型的平台，有助于提升开发人员的开发效率。

参 考 文 献

[1] 贾计东,张明路.人机安全交互技术研究进展及发展趋势[J].机械工程学报,2020,56(03):16-30.

[2] 胡琪,朱定局,吴惠猕,巫丽红.智能推荐系统研究综述[J].计算机系统应用,2022,31(04):47-58.

[3] 孔祥维,唐鑫泽,王子明.人工智能决策可解释性的研究综述[J].系统工程理论与实践,2021,41(02):524-536.

[4] Zhang D, Mishra S, Brynjolfsson E, et al. The AI index 2021 annual report[J]. arXiv preprint arXiv:2103.06312, 2021.

[5] Alshangiti M, Sapkota H, Murukannaiah P K, et al. Why is developing machine learning applications challenging? a study on stack overflow posts[C]//2019 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM). IEEE, 2019: 1-11.

[6] Zhang T, Gao C, Ma L, et al. An empirical study of common challenges in developing deep learning applications[C]//2019 IEEE 30th International Symposium on Software Reliability Engineering (ISSRE). IEEE, 2019: 104-115.

[7] Guo Q, Chen S, Xie X, et al. An empirical study towards characterizing deep learning development and deployment across different frameworks and platforms[C]//2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE). IEEE, 2019: 810-822.

[8] Humbatova N, Jahangirova G, Bavota G, et al. Taxonomy of real faults in deep learning systems[C]//Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering. 2020: 1110-1121.

[9] Islam M J, Pan R, Nguyen G, et al. Repairing deep neural networks: Fix patterns and challenges[C]//2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE). IEEE, 2020: 1135-1146.

[10] Liu M, Peng X, Marcus A, et al. Generating query-specific class API summaries[C]//Proceedings of the 2019 27th ACM joint meeting on European software engineering conference and symposium on the foundations of software engineering. 2019: 120-130.

[11] Wang C, Peng X, Liu M, et al. A learning-based approach for automatic construction of domain glossary from source code and documentation[C]//Proceedings of the 2019 27th ACM joint meeting on european software engineering conference and symposium on the foundations of software engineering. 2019: 97-108.

[12] Li H, Li S, Sun J, et al. Improving api caveats accessibility by mining api caveats knowledge graph[C]//2018 IEEE International Conference on Software Maintenance and Evolution (ICSME). IEEE, 2018: 183-193.

[13] Liu Y, Liu M, Peng X, et al. Generating concept based API element comparison using a knowledge graph[C]//Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering. 2020: 834-845.

[14] 邢双双, 刘名威, 彭鑫. 基于软件知识图谱的代码语义标签自动生成方法[J]. 软件学报, 2021: 0-0.

[15] Prana G A A, Treude C, Thung F, et al. Categorizing the content of github readme files[J]. Empirical Software Engineering, 2019, 24(3): 1296-1327.

[16] Yamada I, Asai A, Sakuma J, et al. Wikipedia2Vec: An Efficient Toolkit for Learning and Visualizing the Embeddings of Words and Entities from Wikipedia[C]//EMNLP (Demos). 2020.

[17] McHugh M L. Interrater reliability: the kappa statistic[J]. Biochemia medica, 2012, 22(3): 276-282.