分　类　号：TP311　　单位代码：10183
研究生学号：2019532053　　密　　级：公 开

吉 林 大 学

硕士学位论文

（学术学位）

NTR：一种基于代码嵌入的 node 包标签推荐方法

NTR：A Node Package Tag Recommendation Method Based on Code Embedding

作者姓名：王昊
专　　业：计算机软件与理论
研究方向：数据驱动的智能化软件工程
指导教师：刘磊
培养单位：计算机科学与技术学院

2022 年 9 月

NTR：一种基于代码嵌入的 node 包标签推荐方法

NTR：A Node Package Tag Recommendation Method Based on Code Embedding

作者姓名：王昊

专业名称：计算机软件与理论

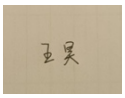
指导教师：刘磊

学位类别：工学硕士

答辩日期： 2022 年 9 月 1 日

吉林大学硕士学位论文原创性声明

本人郑重声明：所呈交学位论文，是本人在指导教师的指导下，独立进行研究工作所取得的成果。除文中已经注明引用的内容外，本论文不包含任何其他个人或集体已经发表或撰写过的作品成果。对本文的研究做出重要贡献的个人和集体，均已在文中以明确方式标明。本人完全意识到本声明的法律结果由本人承担。

学位论文作者签名： 

日期： 2022 年 9 月 1 日

关于学位论文使用授权的声明

本人完全了解吉林大学有关保留、使用学位论文的规定，同意吉林大学保留或向国家有关部门或机构送交论文的复印件和电子版，允许论文被查阅和借阅；本人授权吉林大学可以将本学位论文的全部或部分内容编入有关数据库进行检索，可以采用影印、缩印或其他复制手段保存论文和汇编本学位论文。

（保密论文在解密后应遵守此规定）

论文级别：■硕士 □博士

学科专业： 软件与理论

论文题目： NTR：一种基于代码嵌入的 node 包标签推荐方法

作者签名：  指导教师签名： 

2022 年 9 月 1 日

摘要

NTR：一种基于代码嵌入的 node 包标签推荐方法

作为 Node.js 默认的 JavaScript 语言开源软件包管理系统，截至 2022 年，NPM（Node Package Manager）中存在约 160 万个 Node.js 软件包。开发人员可以基于 NPM 完成 Node.js 软件包的安装、卸载、更新、查看、搜索、发布等操作。其中，NPM 通过匹配搜索关键词与包标签的机制帮助开发人员查找想要使用的 Node.js 包，然而这种匹配机制无法覆盖到没有标签或者被标记得不够完善的开源 Node.js 包，影响搜索结果的可用性。现有的开源软件包标签推荐方法大多采用文本分析技术基于 Readme 文档生成标签，但 Node.js 包的 Readme 文档中往往缺乏详尽的产品描述信息，且无关的噪声信息繁多，使得如何有效地为 Node.js 包添加标签成为了当下的研究难点和挑战。

为此，本文从 Node.js 软件包的源代码出发，提出了一种基于代码嵌入的 Node.js 包标签推荐方法 NTR，包含代码嵌入和多标签分类两个模块。

1）代码嵌入模块旨在学习 Node.js 软件包的代码嵌入向量，主要是要借助函数调用关系来表示源码的语义含义。首先 NTR 借助 Madge 和 ECMAScript 开发工具解析 Node.js 软件包的 JavaScript 源代码，其次，制定 7 条语法规则定位代码抽象语法树中的函数节点来辅助 Node.js 软件包函数调用图的构建。然后利用文本卷积神经网络作为嵌入模型并结合词嵌入技术将 Node.js 软件包的函数调用序列转换为数值向量。

2）多标签分类模块用于为 Node.js 软件包推荐标签。在该模块下，NTR 首先使用 K-means 聚类算法对软件包的标签进行聚类，随后搭建一个多层感知器的神经网络模型进行多标签分类，该模型以 Node.js 包向量为输入，并在输出层进行标签集群的概率预测，NTR 选择概率值靠前的 k 个标签集群作为候选的标签推荐结果。

在实验设计部分，本文选择 EnTagRec 和 tagCNN 标签推荐方法作为对比方法，这两种方法均利用软件包的描述文本进行标签推荐。实验指标是推荐 k 个标签集群时模型的 P 值，R 值和 F 值。实验结果证明在 Node.js 软件包的标签推荐场景下，基于代码嵌入的标签推荐方法 NTR 表现优于基于描述文本的标签推荐方法 EnTagRec 和 tagCNN。NTR 方法在 K 为 50 时，可以取得 Recall@5 值为

0.640，Recall@10 值为 0.768 的表现；在 K 为 100 时，可以取得 Recall@5 值为 0.607，Recall@10 值为 0.715 的表现。

关键词：

标签推荐，深度学习，程序分析，代码嵌入

Abstract

NTR：A Node Package Tag Recommendation Method Based on Code Embedding

As the default JavaScript language open-source package management system for Node.js, as of 2022, there are about 1.6 million Node.js packages in NPM (short for Node Package Manager). Developers can install, uninstall, update, view, search, and publish Node.js packages based on NPM. Among them, NPM helps developers find the Node.js package they want to use by matching search keywords and package tags. However, this matching mechanism cannot cover open-source Node.js packages that have no tags or are not tagged well, which affects the availability of search results. Most of the existing open-source software package tag recommendation methods use text analysis technology to generate tags based on Readme documents. However, the Readme documents of Node.js packages often do not contain detailed product description information, and there is a lot of irrelevant noise information, which makes how to effectively generate tags for Node.js packages become a current research difficulty and challenge.

Therefore, starting from the source code of Node.js package, this paper proposes a Node.js package tag recommendation method NTR based on code embedding, which includes two modules: code embedding and multi-label classification.

1) The code embedding module aims to learn the code embedding vector of the Node.js software package, mainly to express the semantic meaning of the source code by means of the function call relationship. First, NTR uses Madge and ECMAScript development tools to parse the JavaScript source code of the Node.js package. Second, it formulates 7 syntax rules to locate the function node in the code abstract syntax tree to assist the construction of the function call graph of the Node.js package. And then, the text convolutional neural network combined with the word embedding technology is used to convert the function call sequence of the Node.js software package into a numerical vector.

2) The multi-label classification module is used to recommend tags for Node.js packages. Under this module, NTR first uses the K-means clustering algorithm to cluster the tags of the software packages, and then builds a multi-layer perceptron neural network model for multi-label classification. The model takes the Node.js package vector as input, and performs probability prediction of tag clusters in the output layer. Then NTR selects the k tag clusters with the highest probability values as the candidate recommendation results.

In the experimental design part, this paper uses the description text-based methods EnTagRec and tagCNN as the comparison methods. The experimental indicators are selected as the P value, R value and F value of the model when recommending k tag clusters. The experimental results show that in the tag recommendation scenario of Node.js software package, NTR outperforms EnTagRec and tagCNN. And NTR can obtain the performance of Recall@5 value of 0.640 and Recall@10 value of 0.768 when K equals to 50, and can obtain the performance of Recall@5 value of 0.607 and Recall@10 value of 0.715 when K equals to 100.

Keywords:

Tag Recommendation, Deep Learning, Program Analysis, Code Embedding

目 录

第 1 章 绪论

1.1 研究背景与意义

Node.js 是一个基于 Chrome V8 引擎的 JavaScript 运行时的环境，使得 JavaScript 程序可以在服务器端运行。和其他的后端语言开发相比，Node.js 开发具有自己的独特优势，它借助了 JavaScript 语言天生的事件驱动机制同时采用异步非阻塞 IO 技术来支持高性能网页应用的开发，适用于不包含 CPU 密集型操作，但需要处理大量并发 IO 的业务场景，这种特殊的优势使得 Node.js 在后端开发技术中占有一席之地。NPM（Node Package Manager）是 Node.js 默认的 Node.js 软件包的官方管理平台。NPM 平台维护了一个基于 CouchDB 的数据库，该数据库记录了每个 Node.js 软件包的详细信息。同时，开发者也可以基于 NPM 平台完成 Node.js 软件包的安装、卸载、更新、查看、搜索、发布等操作。目前，在 NPM 平台上存在 1,608,527 个 Node.js 软件包，并且这个数字还在不断的增加。

面对如此庞大数量的 Node.js 软件包，开发人员很难快速定位到满足需求的那一个。因此为了简化搜索过程，NPM 平台提供了一种关键字匹配机制，即开发人员通过输入关键词信息，平台就可以推荐以该关键词作为标签的 Node.js 软件包。但是关键词匹配机制无法覆盖到没有标签或者被标记得不够完善的软件项目，从而影响了推荐结果的准确性。为了提高这种以标签作为数据源的信息检索服务的质量，有很多致力于研究如何为实体推荐高质量标签的相关工作[1-7]。这些标签推荐方法大多是从实体的文本特征，譬如：标题，描述或者 Readme 文档等，中抽取出现频率较高的关键词作为候选的推荐标签，或者是利用实体的文本特征作为语料训练模型进行文本分类从而完成标签推荐。然而在 Node.js 软件包的标签推荐场景下，存在包的描述信息不充分且 Readme 文档中噪声信息冗余的问题。从包的描述文本分析，本文统计了 58,753 个 Node.js 软件包的描述文本信息，统计结果显示，832 个包缺少描述信息而且 76.62%的包描述文本长度少于 10 个单词，这意味着描述文本中包含的有效信息太少，不足以充分概括一个包的特点。如果直接使用描述文本进行标签推荐，模型的性能会受到限制。从包的 Readme 文档分析，不难发现 Readme 文档中的信息有很多是在介绍包的安装、配置以及使用说明等等，这种文本信息对于标签推荐而言意义不大，属于噪声信

息，会影响推荐结果的准确性。此外 Readme 文本中还有可能包含一些代码片段信息，这种自然语言和代码混合的文本预处理步骤也会很繁琐。因此在这种场景下，利用描述文本或者 Readme 文档进行标签推荐可能会表现不好，甚至在 Node.js 包缺少描述信息或缺少 Readme 文档时无法正常工作。

除了常用的描述文本和 Readme 文档之外，存在另外一种可以充分地、直观地描述一个 Node.js 软件包功能的文本信息，它就是 Node.js 包的源代码。近年来，结合深度学习的方法进行程序代码分析的相关工作[8-18]层出不穷，深度学习和程序代码分析技术的完美结合也直接促使这样一个想法的形成：是否可以从软件包的源代码文本出发，结合深度学习的方法为 Node.js 软件包进行标签推荐？因此本文提出了一种基于代码嵌入的 Node.js 包标签推荐方法 NTR（Node.js Tag Recommendation），具体的步骤包括：首先需要通过静态程序分析大致构建出了 Node.js 包的函数调用图，然后结合卷积神经网络 CNN（Convolution Neural Network）和词嵌入技术来处理 Node.js 软件包的函数调用序列，为每一个 Node.js 软件包学习到一个可以保留其源代码语义信息的向量表示，最后利用多标签分类模型将 Node.js 包的特征向量分类到不同的标签集群下，从而完成 Node.js 包标签推荐的工作。

1.2 研究现状

NTR 是一种基于代码嵌入的标签推荐方法。本小节将主要围绕典型的标签推荐方法，代码嵌入和程序分析以及多标签分类模型这三个方面研究现状展开介绍。

1.2.1 标签推荐方法

标签是推荐系统中被广泛使用的一种机制，它提供了一种简单的注释方式帮助开发人员快速搜索并且定位到相关的软件项目。大多数开源信息网站都需要依靠标签来管理网站，以提高项目分类，项目推荐等相关操作的准确率和性能。因此，标签的质量对于开源网站而言至关重要。但是，开发人员给软件项目添加标签的过程却是一个主观的行为，不仅依赖于开发人员对项目的理解，也取决于开发人员的语言表达能力和个人偏好。如果缺少自动化的标签推荐工具辅助开发人员给软件项目添加标签，结果就会是随着软件项目的不断增加，标签的数量也会迅速增长，噪声标签增多，从而导致软件项目的管理变得困难。如何为开源网站

上的软件项目推荐标签以及如何推荐高质量的标签成为一个值得被研究的问题。

2010 年，Al-Koafahi 等人基于 IBM Jazz 软件仓库的实证研究发现该网站上有 73%的项目都没有被标记，基于这种标签缺失的现状，他们提出了一种基于模糊集的自动化标签推荐方法 TagRec[1]，这种方法的一大优势就是在于可以根据开发人员的反馈信息及时更新标签的推荐结果。Wang 等人提出了一种组合的自动化标签推荐方法 TagCombine[2],这种方法的结构有三个基本组件组成，第一个组件是多标签分类组件，第二个组件是根据语义相似的软件项目进行标签推荐，第三个组件则是根据单词之间共同出现的频率进行标签推荐，综合这三个组件的推荐结果得到最终的标签推荐列表。武汉大学的周平义等人则是设计实现了一种工具 TagMulRec[3]用于大型软件信息网站的自动化标签推荐，这种方法是根据软件项目描述文本语义的相似性选择候选的推荐标签，然后利用多分类算法对候选标签进行排序完成推荐工作。通过学习被标记良好的软件项目训练模型进行标签推荐也是一种常见的标签推荐手段，这种方法通过为软件项目推荐已有的标签，避免了标签冗余的问题。Wang 等人提出了一种自动化的标签推荐器 EnTagRec[4]，这种方法的主要原理是训练 L-LDA 模型，为输入的软件项目的描述文本序列计算每一个标签主题的概率值。后续他们又对现有方法进行了优化，提出了 EnTagRec 的改进版本 EnTagRec++[5]。和只处理软件项目描述文本信息的 EnTagRec 不一样，EnTagRec++在为软件项目推荐标签时会参考软件项目作者其他项目的标签，他们认为软件项目的作者往往会对一个固定的领域感兴趣，因此同一个作者推荐的标签往往是相似的。除了这些传统的标签推荐方法之外，也可以将标签推荐的问题抽象成文本的分类问题，结合深度学习进行回归预测。2019 年，周平义的团队验证了这样一个问题：在软件信息网站的标签推荐任务中，深度学习方法是否会比传统的方法好？在该文章[6]中，他们把标签推荐问题当成回归问题处理，并且设计实现了 TagCNN，TagRNN 以及 TagHNN 等多种不同的网络结构进行标签推荐，实验结果证明使用适当的深度学习方法能够获得比传统方法更好的性能。同样也是深度学习的方法，Liu 等人设计实现了一种加入注意力机制的胶囊神经网络的结构[7]，这种方法通过完成分类任务实现为文章推荐标签的目的。

这些基于文本分类的标签推荐方法通常利用实体的描述文本作为数据源训

练模型，但是在 Node.js 软件包的标签推荐场景下，描述文本往往过短，包含的信息有限，不适合单独作为标签推荐模型的数据源。而如果直接使用 Readme 文档作为文本特征进行分类的话，不仅预处理步骤繁琐而且 Readme 文档中的噪声信息太多，影响推荐结果的准确性。因此考虑到了这些局限性，NTR 方法选择从 Node.js 软件包的源代码文本出发，通过实现对 Node.js 包代码特征向量的多标签分类完成标签推荐的任务。

1.2.2 代码嵌入与程序分析

Embedding 是一种学习实体向量表示的技术，也被称作嵌入技术，这种技术能够使相似实体对应的向量表示在向量空间中彼此相邻。代码嵌入（code embedding）即学习代码向量表示的技术[8]，使得语义相似的代码信息有相似的向量表示。

随着深度学习技术在自然语言处理领域被广泛使用之后，研究人员逐渐把注意力投向了程序源代码这种具有特殊结构的文本信息上。如何利用深度学习的技术进行程序的分析成为一个研究的热点问题。代码嵌入工作的研究就是其中的一个重要分支。早在 2014 年，Peng 的团队最早提出了利用深度学习技术进行程序分析的想法，他们制定了“code criterion”[9]这种计算方式来为代码抽象语法树中的每一个节点学习对应的向量表示。基于“code criterion”，次年他们又发表一篇论文，提出了一种专门用于处理抽象语法树的卷积神经网络 TBCNN[10]来捕获代码片段的特征，分类实验结果的高准确率证明了他们的模型的确能够捕获代码片段的特征,准确地学习到代码片段的向量表示。在这之后，有很多工作都在研究如何设计模型能够更加准确的抽取代码特征，学习代码的向量表示，这一类利用神经网络捕获代码特征，学习能够保留代码片段语义信息的向量表示的工作都可以称作代码嵌入。Alon 团队是提出了一种基于抽象语法树路径的方法处理程序代码[11]，这种基于路径的处理方法最大的优势在于能够最大限度的保留抽象语法树的结构信息，在这种基于抽象语法树路径的处理方法之上，2019 年他们又提出一种加入了注意力机制的神经网络 code2vec[12]进行代码嵌入的工作，其中注意力向量的作用就是为不同的路径分配不同的权重。Zhang 设计了一个基于抽象语法树的神经网络 ASTNN[13],这种方法将整个抽象语法树拆分为一系列的子树分开处理，通过处理每一棵子树以生成代码片段最终的向量表示结果。除了函数粒

度的分析工作之外，也有很多其他粒度的程序分析工作。比如在包粒度的分析工作中，Mateless.等人提出了一种分级神经网络的方法 Pkg2Vec[14]，为安卓应用的软件包 APK 文件学习向量表示，解决代码作者归属的问题。这种方法把一个完整的 APK 文件当作是函数的集合，通过处理每一个函数中的关键词以及 API 的调用来完成包的嵌入工作。在单词（token）粒度的分析工作中，Bart.等人提出了名为 Import2vec[15]的工具，他们通过分析实际工程项目的导入语句，抽取出软件库的使用文本序列，然后借鉴词向量的训练思想，利用这些抽取到的文本序列作为训练的语料库训练 skip-gram 模型，为软件库的名称学习词向量，完成后续的分析工作。

当然，深度学习的技术也可以应用到其他代码分析相关的应用场景，解决具体的实际问题。比如，Liu Kui[16]的团队利用卷积神经网络 CNN 抽取函数代码片段的向量特征，通过向量之间的相似性进行函数名称的一致性检测。同样是处理函数名称相关的工作，Son 等人则是把函数名称推荐的工作当成机器学习三大任务中的生成任务，他们提出了一种 MNIRE[17]的方法，这种方法的底层是基于一个 seq2seq 网络模型，模型的输入为代码片段序列输出为函数名称序列的。Rabee 等人构建了一种基于 LSTM 的网络模型 NL2Type[18],该网络模型通过输入函数体的代码文本信息，输出层利用 softmax 函数进行回归预测，从而完成 JavaScript 函数返回值类型的推断任务。

在这些代码嵌入工作的启发下，本文也希望能够通过深度学习模型得到 Node.js 软件包的代码嵌入。和函数粒度的代码嵌入工作相比，Node.js 软件包级别的代码嵌入更加抽象。为了获取到软件包级别的训练文本，本文通过静态程序分析的技术构建出了软件包的函数调用图，并且利用卷积神经网络作为嵌入模型结合词嵌入技术将 Node.js 软件包的函数调用序列转换成了数值向量。

1.2.3 多标签分类方法

分类任务是机器学习的重要任务，就是将实体划分到一组不相交标签集合中的某一个标签下，如果标签集合中只有两个标签，则为二分类问题，如果标签集合中标签的个数大于二，则称为多分类问题。相应的，多标签分类问题就是将实体划分到标签集合中的多个标签下。

在关于多标签分类问题的这篇综述中[19]，Grigoris 对比了不同的多标签分类

方法，并将现有的多标签分类算法按照处理手段的不同分成了两大类，分别是问题转换方法以及算法适应方法。问题转换方法就是将多标签的分类问题转换为普通的分类问题；算法适应方法就是指通过扩展特定的学习算法来直接处理多标签数据的方法。经典的多标签分类算法 ML-KNN[20]就是一种算法适应方法，ML-KNN 的主要思想通过寻找临近的 K 个样本，利用贝叶斯条件概率，为当前样本计算每一个标签为 0 或者 1 的概率。而随着深度学习技术在人工智能领域的突破性进展，训练神经网络模型也逐渐成为解决多标签分类任务的热门方法。Yang 等人就提出了加入了注意力机制的生成网络模型 SGM[21]用于文本的多标签分类任务,这种方法考虑了标签之间的相互关系，此外模型在预测时，注意力机制的引入使得输入的不同部分对预测结果存在不同程度的影响。Thibaut[22]等人则是提出一种专门用于解决在标签缺失的场景下如何为图片进行多标签分类，预测缺失标签的深度学习模型。

NTR 方法将标签推荐任务当作多标签分类任务处理。为了实现给输入样本推荐 k 个最相关的标签，本文选择使用深度神经网络作为多标签分类模型，因为和其他直接将输入样本分到不同类别下的方法不一样，深度神经网络可以为输入样本选择出概率值靠前的标签集群作为推荐结果。

1.3 本文主要工作

本文提出了一种基于代码嵌入的 Node.js 包标签推荐方法 NTR，该方法利用卷积神经网络抽取 Node.js 软件包源代码的特征向量，并结合多标签分类模型对这些特征向量进行分类，从而实现 Node.js 软件包的标签推荐。具体的步骤如下：首先使用 Madge 和 ECMAScript 开发工具解析 Node.js 软件包的 JavaScript 源代码，提取出包中各文件模块之间的函数调用关系，基于这些调用关系大致构建出包的函数调用图，并通过图的深度优先遍历得到包的函数调用序列；紧接着，利用卷积神经网络作为嵌入模型并且结合词嵌入技术将预处理过的函数调用序列转换为数值向量；最后，通过监督学习训练多标签分类模型对 Node.js 软件包特征向量进行分类，从而完成标签推荐的工作。在多标签分类的过程中，需要声明的是本文事先根据标签的词向量对标签进行了聚类，所有的标签被划分到了 K 个不同的标签集群下，且同一集群下的标签语义基本相似。对于需要被推荐标签的 Node.js 软件包而言，多标签分类模型将会为其特征向量计算出各个标签集群的

概率，概率值靠前的 k 个标签集群下的标签将会作为最终的推荐结果（K 和 k 的实际含义不一样，K 代表标签集群的总数量，k 为推荐的标签集群数量）。

为了验证 NTR 的有效性，在实验环节部分本文提出并且讨论了如下的两个研究问题：

1) 本文提出的嵌入方法是否可以准确学习到 Node.js 软件包源代码的向量化表示？
2) 本文提出的标签推荐方法 NTR 是否可以合理地为 Node.js 软件包推荐标签？

实现结果表明，NTR 可以准确地为 Node.js 软件包学习到保留源代码语义信息的向量化表示。同时在这种描述文本简短甚至缺失的标签推荐场景下，基于代码嵌入的标签推荐方法 NTR 表现优于基于描述文本的标签推荐方法 EnTagRec[5] 和 tagCNN[6]。

本文工作的具体贡献如下：

1) 本文提出了一种可以用于学习 Node.js 软件包源代码特征向量的嵌入方法，并且证明这种嵌入方法学习到的向量可以保留代码的语义信息；
2) 本文提出了一种基于代码嵌入的标签推荐方法 NTR 为 Node.js 软件包进行标签推荐；
3) 本文提出的 Node.js 软件包源代码的嵌入方法为开发者在 NPM 中执行一些子任务提供了技术支持，比如 Node.js 软件包的搜索任务以及分类任务等。

1.4 本文组织结构

本文主要包括五个章节，分别为：绪论，相关技术介绍，Node.js 软件包标签推荐，实验部分以及总结与展望部分。每一章内容大致概括如下：

第 1 章首先介绍了本文工作的研究背景以及方法的创新性；然后介绍了标签推荐以及程序分析结合深度学习领域相关的研究工作；最后总结了本文的主要工作和贡献；

第 2 章介绍了本文方法中用到的技术和理论，主要包括 Node.js 开发相关的理论知识，实体嵌入技术的应用以及深度神经网络相关的基础知识等；

第 3 章介绍了本文方法的流程和细节，具体包括：如何利用程序静态分析构

建 Node.js 软件包的函数调用图，如何利用卷积神经网络作为嵌入模型学习 Node.js 软件包源代码的向量表示以及如何利用多标签分类模型为 Node.js 软件包推荐标签；

第 4 章为本文的实验环节，该部分主要用于验证两个研究问题：“本文提出的嵌入方法是否可以准确学习到 Node.js 软件包源代码的向量化表示？”以及“本文提出的标签方法 NTR 是否可以合理地为 Node.js 软件包推荐标签？”。

第 5 章为总结与展望部分，概述了本文的所有工作内容，并对 NTR 方法的优化改进提出来解决方案。

第 2 章 相关技术介绍

本章将围绕 NTR 方法中使用到的技术以及相关的理论基础知识展开。这些内容包括了 Node.js 开发的编程特点以及 Node.js 软件包代码分析常用工具的介绍，实体嵌入技术原理的介绍，深度神经网络的基本原理以及常见网络模型的介绍，和常用数据降维方法的介绍。基于这些理论和技术，NTR 实现了 Node.js 软件包的代码嵌入和标签推荐工作。

2.1 Node.js 开发

Node.js 最早是由瑞安.达尔提出来的基于 JavaScript 语言和 Chrome V8 引擎的开源项目，旨在利用 JavaScript 语言的事件驱动机制和非阻塞 IO 的特性来开发高性能的 web 应用，使得 JavaScript 语言也可以在服务器端运行。在 Node.js 的开发环境下，开发者采用模块化编程来构建 Node.js 应用。这样，一个 Node.js 包可以由多个文件模块组成，每个模块实现一个指定的功能，模块组合起来共同实现一个 Node.js 包完整的功能。尽管一个 Node.js 包可能是由多个文件模块组成，但并非包下的所有文件模块都与包的功能直接相关，其中可能会存在一些测试文件或者配置文件等这些与包的功能没有直接关系的文件模块，而那些和包的功能直接相关的模块是被主模块（或者称为入口文件模块）依赖的模块，这是因为 Node.js 包的 API 会在主模块中声明导出，被主模块依赖意味着主模块调用了依赖模块中的代码。主模块的路径会声明在配置文件的 main 属性当中，默认值为根目录下的 index 文件。

本文试图通过解析 Node.js 包的源代码，大致构建出包的函数调用图。为了减小静态分析的难度和规模，NTR 从主模块开始分析且只专注于分析被主模块依赖的模块，因为只有这些模块才是和 Node.js 包的功能相关。为了完成构建 Node.js 软件包函数调用图的目标，使用了三个开发工具：Madge[1], Esprima[2]以及 Estraverse[3]。Madge 是一个用于生成 Node.js 项目模块依赖关系可视化图的开发工具，可以利用该工具分析出被主模块依赖的模块。对于 Node.js 包下单个模块的分析而言，Esprima 可以对 JavaScript 程序进行词法分析和语法分析，生成对

[1] https://github.com/pahen/madge
[2] https://esprima.org
[3] https://github.com/estools/estraverse

应的抽象语法树，Estraverse 则是用于 JavaScript 程序抽象语法树的遍历。

2.2 实体嵌入技术

Embedding 是一种学习实体向量表示的技术，也被称作嵌入技术，这种技术能够使相似实体对应的向量表示在向量空间中彼此相邻。

Word2vec[23]是一项为大型数据集中的单词学习连续向量表示的技术。这种技术有两种不同的实现模型：CBOW 模型和 Skip-Gram 模型。其中 CBOW 模型是根据窗口内单词的上下文来预测当前单词进行模型的训练，Skip-Gram 模型则相反，它是根据当前单词来预测窗口内的上下文进行训练，其实本质都是通过捕捉单词与其上下文之间的关系来训练神经网络模型，根据权重矩阵得到每一个单词对应的词向量。

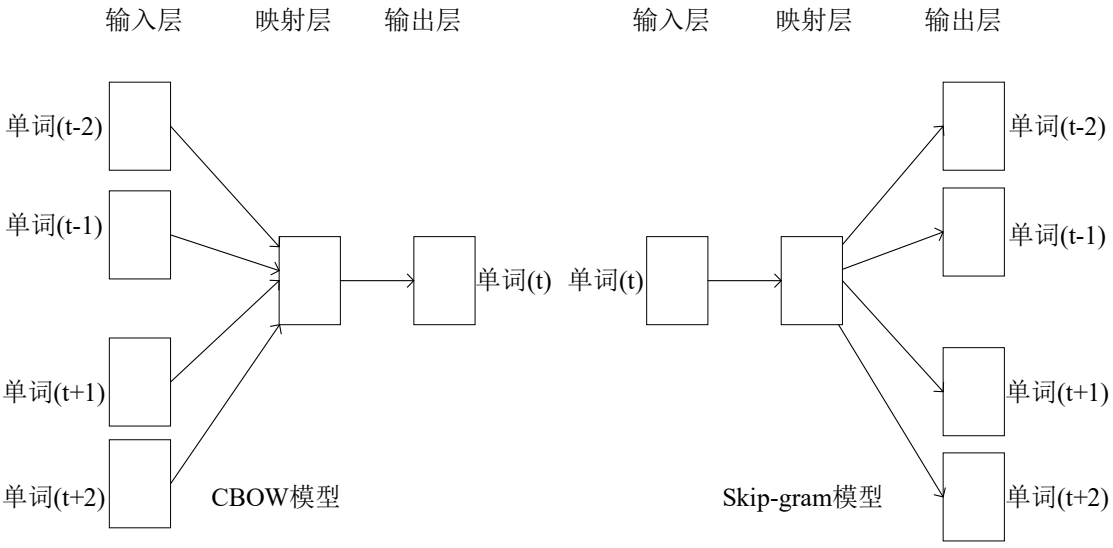


图 2.1　CBOW 模型和 Skip-gram 模型

这种词嵌入技术的好处不仅仅是能够为每一个单词学习到一个固定长度的词向量，避免了 one-hot 编码过于稀疏，存储效率不高的问题，另外一个优势就是相似的单词具有相似的词向量，单词之间的语义关系可以通过计算词向量之间的欧式距离或者余弦相似度来确定。后续地，Word2vec 这两种模型的输出层采用了哈夫曼树的结构取代了原来隐藏层到输出层的 Softmax 映射，这样做的优势在于结合单词的哈夫曼编码（01 组成的字符串）进行一连串的二分类，可以避免输出层对所有的单词进行 Softmax 概率计算，提高了模型的计算效率。另外一种词嵌入技术 fastText[24]则是通过字词算法对长单词进行拆分解决 OOV[25]（Out Of Vocabulary）问题，从而实现对现有词嵌入技术的优化改进。

在本文中，NTR 利用词嵌入技术 word2vec 的目的在于为 Node.js 包的函数调用序列中的 token 学习定长的词向量，确保相似的 token 具有相似的向量表示。通过这些 token 词向量的连接，将 Node.js 包的函数调用序列转换成卷积神经网络 CNN 能够处理的固定大小的类似于灰度图片的二维矩阵。

随着词嵌入技术取得重大突破，类似的嵌入思想也被引入到了程序源代码这种特殊文本实体的处理上。和词嵌入技术类似的是，代码嵌入技术同样期待相似的代码文本具有相似的向量表示结果，所不同的是，代码嵌入技术可以根据不同的使用场景划分为不同的粒度的代码嵌入[26]，比如单词粒度的代码嵌入可以用于代码的自动补全或者修复，函数粒度的代码嵌入可以用于代码克隆检测以及代码片段推荐等。Code2vec[12]是一种函数粒度的代码嵌入方法，它从函数体源代码解析出的抽象语法树中抽取路径（一条路径是由树中任意两个叶子节点以及二者之间的所有非叶子节点组成），将这些路径的文本向量作为模型输入组合得到代码片段的向量表示。Pkg2vec[14]是一种包粒度的代码嵌入方法，该嵌入方法为安卓应用软件包 APK 文件的源代码学习向量表示，解决代码作者归属的问题。这种方法把一个完整的 APK 文件视为是函数的集合，抽取每一个函数中的关键词以及 API 调用相关的信息作为模型输入来完成安卓软件包源代码的嵌入工作。不难看出，代码嵌入技术需要从代码的抽象语法树或者直接从代码文本中抽取有效信息用作嵌入模型的训练语料。

本文的工作的意义之一就是为 Node.js 软件包学习特征向量，完成 Node.js 包源代码的嵌入工作，使得相似功能的 Node.js 软件包应当具有相似的向量表示，为后续的分支工作奠定基础。参照其他代码嵌入技术，包粒度的代码嵌入需要包级别的代码文本信息作为模型的训练语料，因此本文试图通过静态程序分析构建 Node.js 包的函数调用图，把 Node.js 包的函数调用序列文本作为嵌入模型的训练资料。

2.3 深度神经网络

深度神经网络（DNN：deep neural network）是一个包含了多个隐藏层的深度学习框架。凭借着其优越的性能和高准确率，深度神经网络已经被广泛应用于诸多的人工智能的任务场景中，比如图片识别，语言识别以及自动驾驶等等，这是因为深度神经网络在对大量数据进行统计学习之后可以准确地抽取出原始数据

的特征[27]。为了满足不同使用场景，深度神经网络往往具有不同的形态。最早的多层感知器 MLP（multi-layers processor）就是一种简单的深度神经网络。多层感知器的每一层都全连接到下一层，通过将一组输入向量映射到一组输出向量上进行数据预测。

卷积神经网络 CNN 是一种常见形式的深度神经网络，而 LeNet-5[28]则是卷积神经网络的经典实现，它由 7 层网络结构组成，模型的输入是固定大小的灰度图片，前 5 层由交替的卷积层和子采样层组成，用于图片特征的捕获，后 3 层全连接，组成了传统的多层感知器用于最终的分类，该模型能够准确地捕获输入图片的特征，完成手写字母和数字的识别任务。除了在图片识别领域，卷积神经网络也逐渐被引入到了自然语言处理相关的任务中，Kim[29]等人设计了一种结合了当时流行的词嵌入技术的卷积神经网络 textCNN 用于电影评论的情感分类，证实了卷积神经网络也能够捕获自然语言的语义特征。循环神经网络 RNN 是另一种常见形式的深度神经网络，它专门用于处理序列数据（一串相互依赖的数据流），在文本生成，机器翻译等应用场景下发挥着至关重要的作用。

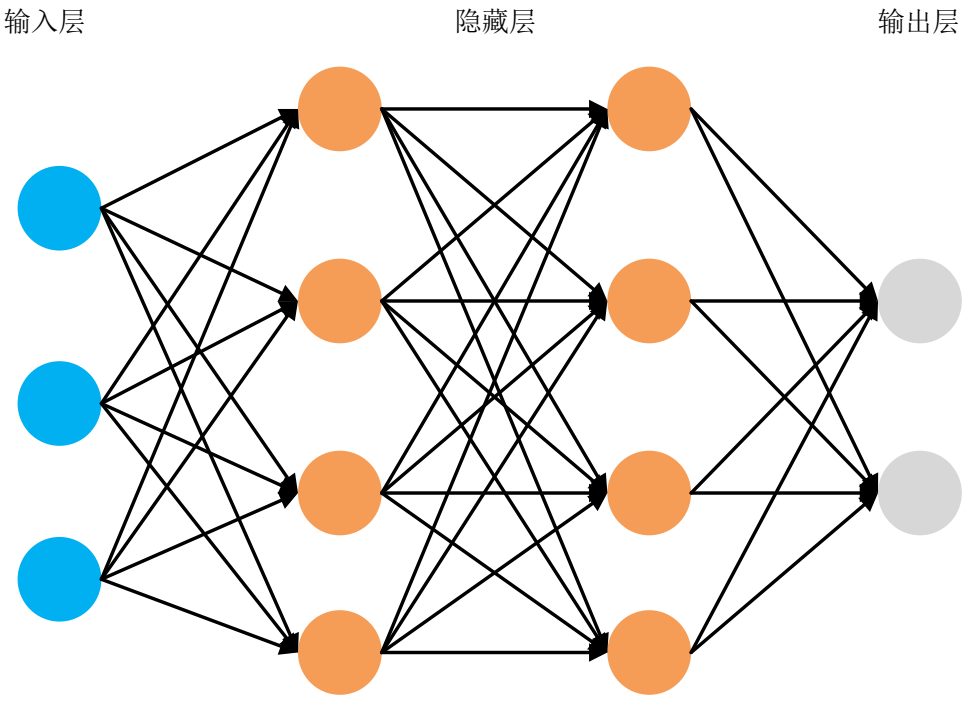


图 2.2　隐藏层数为 2 的多层感知器

本文使用了两种不同结构的深度神经网络：卷积神经网络和多层感知器（包含两个隐藏层）。受到卷积神经网络可以捕获自然语言语义特征的启发，本文将

Node.js 包的源代码视作一种特殊的自然语言，根据 LeNet-5 的模型结构和 textCNN 处理文本的思想，NTR 通过卷积神经网络抽取 Node.js 包源代码的语义特征，学习代码的向量表示。此外，为了完成 Node.js 包的标签推荐工作，通过监督学习训练了一个用于多标签分类的多层感知器。感知器输出层的神经元利用 sigmoid 激活函数计算对应标签集群的概率。

2.4 数据降维

数据降维是指通过某种映射方法，将高纬的特征映射到低维的数据空间。上文提到的实体嵌入技术用于学习实体的向量表示，这种学习到的实体向量往往需要通过数据降维，降低到 2 或则 3 维度，然后通过可视化去直观地观察向量之间的邻近关系，从而衡量嵌入技术的好坏。常用的降维算法有主成分分析 PCA[30]（Principal Component Analysis），t 分布随机领域嵌入算法 t-SNE[31]（t-distributed Stochastic Neighbor Embedding）等。

PCA 是一种线性的降维方法，其主要思想是利用正交变换实现数据从高纬到低维的变换，从而在更小的维度空间下展示数据的主要特征。这种方法的局限性在于无法解释特征之间复杂的多项式关系，不能够同时保留数据的局部和全局结构。t-SNE 比 PCA 适用于高纬特征的降维可视化，是对降维方法 SNE[32]的改进。其主要思想是将欧式距离转换为条件概率来描 述特征点之间的相似关系。在高纬空间下构建特征点之间的概率分布，在低纬空间下使用 t 分布取代高斯分布构建点之间的概率分布，通过训练不断优化两个概率分布之间的 KL 散度更新目标结果。

本文在实验环节部分利用了 t-SNE 方法对 Node.js 软件包的代码嵌入进行了降维，观察不同类别下 Node.js 软件包代码的嵌入在二维空间下的分布，并将特征的分布情况作为衡量 NTR 嵌入方法的一种指标。

第 3 章 Node.js 软件包标签推荐

针对 Node.js 软件包标签推荐场景下 Readme 文本噪声信息过多，软件包描述文本缺失的问题，本文提出了一种基于代码嵌入的标签推荐方法 NTR。如图 3.1 所示，NTR 的整体结构主要由 Node.js 软件包的嵌入模块和多标签分类模块组成。嵌入模块进行 Node.js 包程序的分析和嵌入模型的训练，多标签分类模块进行标签推荐。

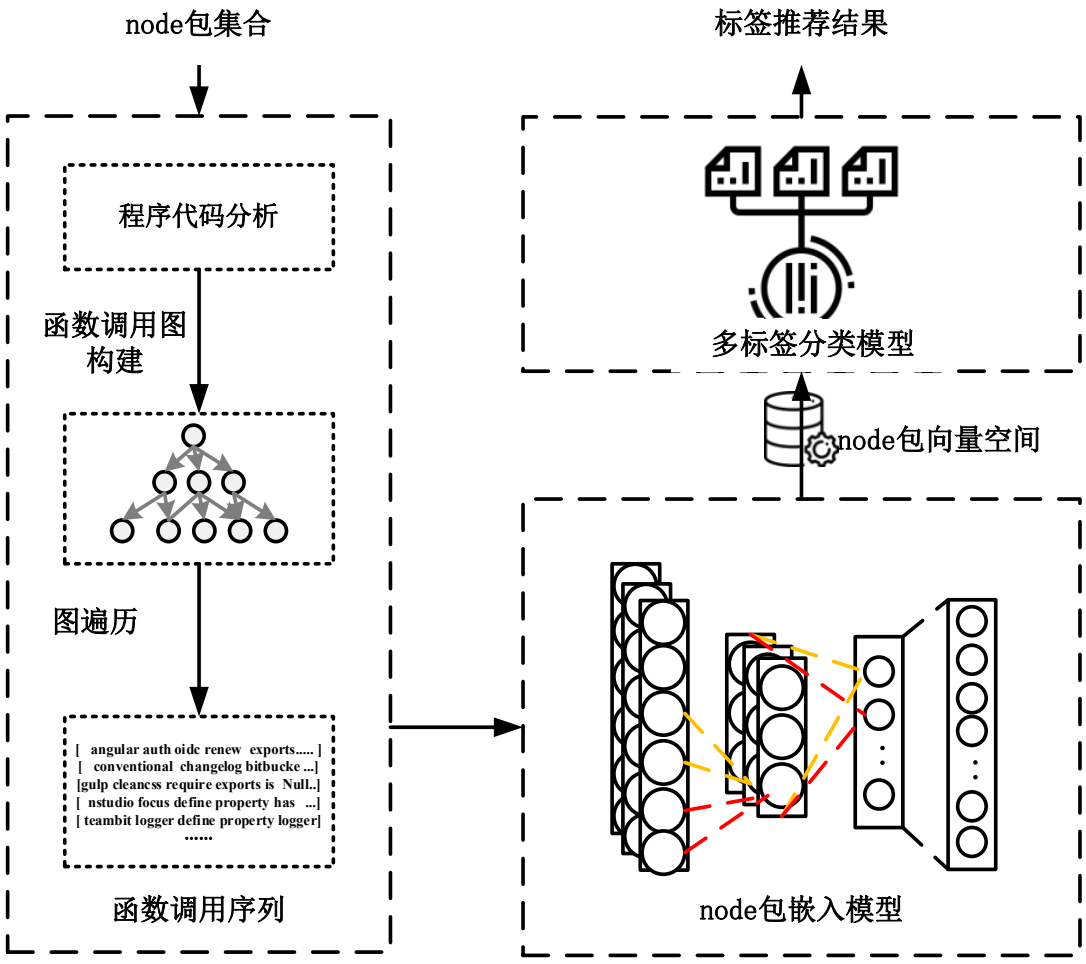


图 3.1 NTR 方法的整体结构

Node.js 软件包的嵌入模块旨在训练嵌入模型学习软件包的代码向量用作分类模型的输入。在该模块下，NTR 首先利用了静态程序分析的技术解析包的 JavaScript 源代码，通过制定语法规则定位抽象语法树中函数节点构造出包的函数调用图。然后利用文本卷积神经网络作为包的嵌入模型，同时结合词嵌入技术将 Node.js 包的函数调用序列转换为数值向量，完成 Node.js 包的嵌入。多标签分类模块用于软件包的标签推荐。在该模块下，NTR 首先利用聚类算法将软件

标签划分到了不同的标签集群下，随后搭建了一个多层感知器的神经网络模型，该模型以 Node.js 包向量为输入并在输出层计算不同标签集群的概率，选择概率值靠前的 k 个标签集群作为 NTR 方法的推荐结果。接下来，本章将分为三个小节来详细介绍 NTR 的具体实现。

3.1 函数调用关系抽取

函数粒度的代码嵌入工作是将代码解析成抽象语法树，然后从语法树结构中抽取嵌入模型所需的训练语料。然而 Node.js 包的模块化编程机制却阻碍了直接采用基于抽象语法树的代码嵌入方法来嵌入 Node.js 软件包，因为这种基于抽象语法树的代码嵌入方法忽略了包模块之间的相互调用关系。一个 Node.js 包是由多个文件模块组成，每个模块都可以直接被解析成一个独立的抽象语法树，如果直接采用遍历抽象语法树的代码嵌入方法，必然会涉及到树的合并和剪枝。而 Node.js 包的函数调用图可以很好地反映出包的功能特性，它是一种控制流图用以展示计算机程序中函数之间的相互调用关系。因此本文利用静态程序分析的技术构建 Node.js 包的函数调用图，通过遍历函数调用图收集软件包的函数调用关系序列，并将该文本序列用作包嵌入模型的训练语料。

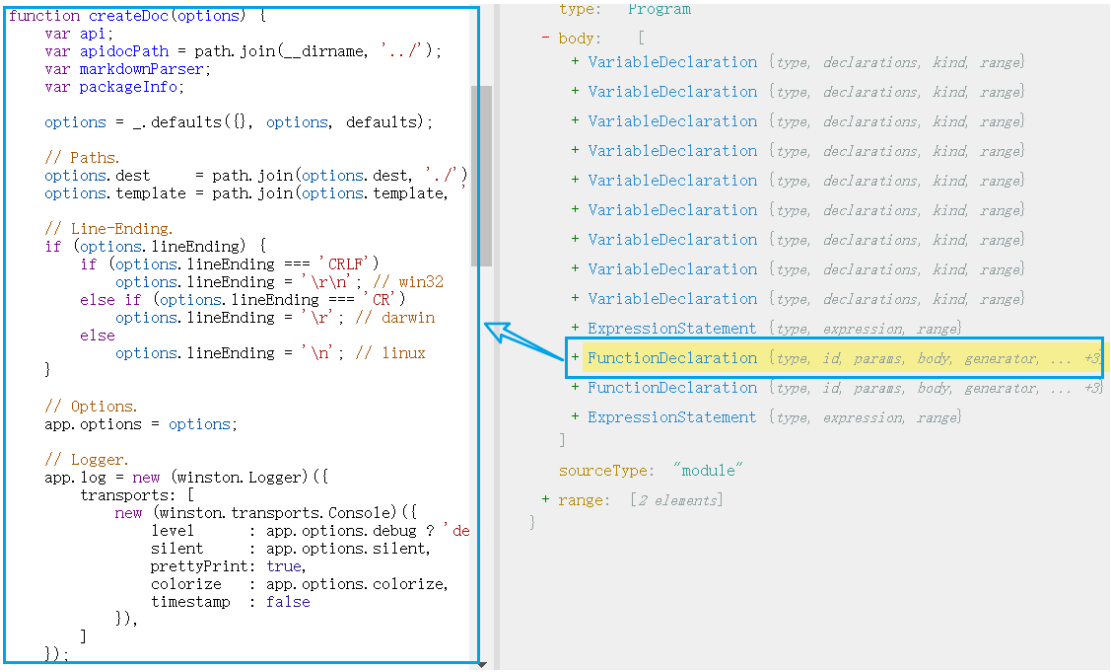


图 3.2 Apidoc-ui 包下 index.js 模块语法树结构

3.1.1 抽象语法树构建

Node.js 软件包函数调用图的构建需要基于包下每一个文本模块内部的函数

名称信息和函数调用信息来实现。这些文本信息是从文件模块的抽象语法树上抽取到的。Esprima 是一个高性能的 JavaScript 程序的解析器，它支持将 Node.js 包下每一个文件模块解析成对应的抽象语法树结构。图 3.2 展示了 apidoc-ui 软件包下主模块 index.js 文件的抽象语法树结构。如图所示，模块中的代码块被解析成了抽象语法树上不同类型的节点，树上每个节点都是一个 JavaScript 对象，包含不同的属性。比如，对于树上的第一个函数声明类型的节点而言，其 id 属性就包含了函数的名称信息。

3.1.2 函数节点定位

在模块的抽象语法树构建出来之后，需要定位树上的函数节点，抽取出包内各个模块下函数的名称信息和函数的调用信息来辅助 Node.js 包函数调用图的构建。为了完成抽象语法树上函数节点的定位，本文参考了 ECMAScript[4]语法规范制定了如表 3.1 所示的 7 条规则，用于抽象语法树中函数定义语句以及函数调用语句节点的定位，定位成功之后就可以抽取到函数的名称信息以及其所属的模块信息。

现针对表中的 7 条规则作如下解释。在 JavaScript 中，定义一个新的函数，可以利用 function 关键字进行函数声明，也可以利用函数表达式或者 Function 构造函数的形式把定义的新函数赋值给一个变量或者赋值给对象的一个属性，在调用该函数时可以直接使用变量的名称或者属性的名称进行调用。规则①，对于普通的函数声明节点，节点的名称即位函数名称；规则②，若在定义新的函数时，将函数表达式赋值给一个变量，那么变量的名称就是函数名称，抽取的信息为变量名；规则③，若在定义新的函数时，将函数表达式赋值给某对象的一个属性，那么对象属性的名称为函数名称，抽取的信息为对象属性的名称；规则④，若在定义新的函数时，使用 Function 构造函数，将函数赋值给一个变量，抽取的信息为变量名；规则⑤，若在定义新的函数时，使用 Function 构造函数，将函数赋值给某对象的一个属性，那么对象属性的名称为函数名称，抽取的信息为对象属性的名称。

与函数定义语句节点的定位相比，函数调用语句节点的定位相对简单。JavaScript 中函数调用的形式也是直接调用函数的名称，如果函数属于某个对象，

[4] https://262.ecma-international.org/12.0/#sec-intro

则需要通过对象访问来完成调用，对应于规则⑦；如果不是，直接使用函数名称进行调用，对应于规则⑥。

表 3.1　语法树函数节点定位规则

规则	语法树函数节点定位规则	抽取信息
规则①	node.type=FuntionDeclaration	node.id.name
规则②	node.type=VariableDeclaration node.init.type=FunctionExpression	node.id.name
规则③	node.type=ExpressionStatement node.expression.right.type=FunctionExpression	node.expression.left.property.name
规则④	node.type=VariableDeclaration node.init.type=NewExpression node.init.callee.name=Function	node.id.name
规则⑤	node type=ExpressionStatement node.expression.right.type=NewExpressionode.expression.right.callee.name=Function	node.expression.left.property.name
规则⑥	node.type=ExpressionStatement node.expression.type=callExpression	node.expression. callee.name
规则⑦	node.type=ExpressionStatement node.expression.type=callExpression	node.expression. callee.property.name

3.1.3 函数调用图构建

为了更好地理解本文方法中提及的函数调用图，本文参考了 AppDNA[33]的工作，对 Node.js 软件包的函数调用图做了如下定义：给定一个 Node.js 软件包，它的函数调用图是一个有向图 G=(N, E, L)，其中：

1. 顶点集 N 是一组节点，其中一个节点 n ∈ N 表示 Node.js 包中的某文件模块中的一个函数对象；

2. 边集 E 描述了函数之间的调用关系。从节点 n1 到节点 n2 的边 e ∈ E 表示节点 n1 中的函数 f1 调用了节点 n2 中的函数 f2；

3. L 集是 G 中所有节点的标签集。对于图 G 中的节点 n，其标签由节点 n 中函数对象的名称以及函数对象所属模块的名称组成；

在 Node.js 软件包函数调用图的构建过程中，NTR 首先会为每一个调用图生成一个入口节点，入口节点的标签信息为软件包的名称。随后按照借助于 Madge 工具确立的模块分析顺序依次解析每一个文件模块，每当通过上述规则定位到一个函数结点时，只需要判断该函数节点是否已经存在于图中。如果集合 N 中不包含该函数节点，则在 N 中添加该函数节点，并在集合 L 中添加一条入口节点到该节点的边；如果集合 N 中包含该函数节点，只需要将该函数内的调用关系添加到边集合 L 中。由于不同的文件模块中可能会包含同名函数，如果不加以区分就会导致构建的函数调用图中部分调用关系出现错乱。为了解决这个问题，在抽取函数信息的同时也需要保存函数所属的模块信息，函数名称和函数所属的模块可以唯一确定一个函数。对于函数定义节点而言，所属模块即为当前分析的模块，对于函数调用节点而言，通过分析代码的“require”语句就可以判断出该函数是本模块的函数还是导入模块的函数。在完成所有功能模块的解析后，软件包的函数调用图构建结束。图 3.3 展示了 Node.js 软件包 apidoc-ui 的函数调用图。

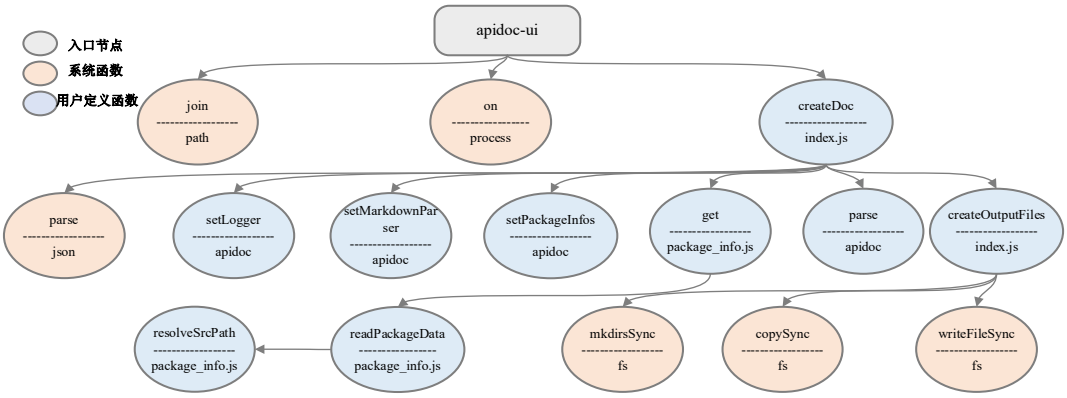


图 3.3　Node.js 包 apidoc 的函数调用图

为了得到包嵌入模型的训练语料，还需要通过图的深度优先遍历获取到每一个 Node.js 软件包的函数调用序列，其中序列中的每个单词对应着包中的一个函数名称，然后根据驼峰命名约定对序列中的每个单词进行分词，并将所有字符转换为小写得到最终的函数调用序列。这些文本序列将作为卷积神经网络模型的训练语料，用来完成 Node.js 软件包的嵌入。经过图的遍历和预处理之后，软件包 apidoc-ui 的函数调用图将被转换成包含语义信息的函数调用关系序列，表 3.2 展

示了该序列的信息。

表 3.2　软件包 apidoc-ui 的函数调用序列

join path on process create doc parse json set logger set markdown parser set package infos get read package data resolve src path parse create outputfiles mkdirs sync fs copy sync fs write file sync fs

3.2 Node.js 软件包嵌入方法

在 Kim 的论文[29]中，他们提出使用 textCNN 的网络结构抽取文本句子的特征进行电影评论的情感分析。同样在论文[16,34]的工作中，Liu 等人利用卷积神经网络模型抽取代码片段的特征向量完成函数名称一致性检测和推荐的工作。这些先前工作的实验结果证明了卷积神经网络模型是可以完成文本特征的抽取，因此 NTR 同样也选择了使用卷积神经网络模型来处理 Node.js 包的函数调用序列，实现 Node.js 包的嵌入工作。在本文中，卷积神经网络模型主要由卷积层，子采样层以及全连接层组成。其中，卷积层和子采样层用于完成输入文本局部特征的捕获，全连接层用于完成局部特征的组合和压缩，生成 Node.js 包的特征向量。图 3.4 描绘了 NTR 中使用到的卷积神经网络模型的基本架构。需要注意的是，本文训练卷积神经网络模型不是为了完成某一项具体的分类任务，只是为了抽取函数调用序列的文本特征，得到 Node.js 包的向量表示结果用于后续的标签推荐工作，因此整个训练过程是一个无监督的学习过程。

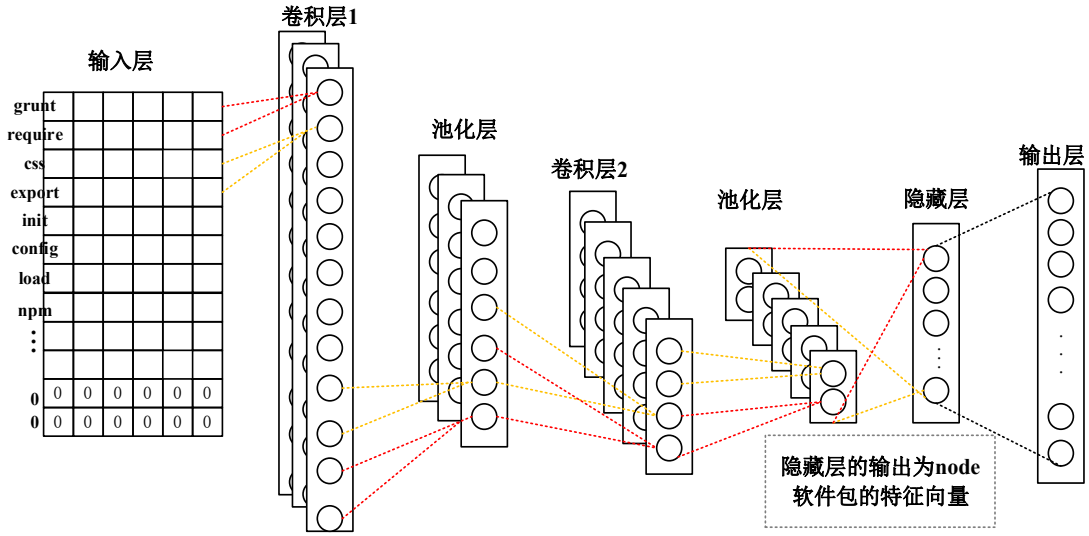


图 3.4　卷积神经网络的结构

3.2.1 token 嵌入

卷积神经模型被广泛应用于图片处理的相关工作中，而图片的本质是像素矩阵。那么，要将 Node.js 包的函数调用序列作为卷积神经模型的输入，必须将该序列表示成模型能够处理的二维数值向量的形式。为了实现这样的转换，本文采用了词嵌入的技术学习输入序列中每一个 token 的词向量，并将这些词向量连接成二维向量。即对于一个 Node.js 包的函数调用序列 $S = (t1, t2, t3, \dots, tn)$，本文利用词嵌入技术学习 S 中的每个单词$ti(\ i\ \in\ [1, n]\ )$的词向量，并将这些词向量连接形成一个二维向量，作为卷积神经网络模型的输入。如图 3.4 所示，输入矩阵中的每一行均为输入序列中每一个单词ti对应的词向量。

因此在 token 的嵌入阶段，需要训练一个词嵌入模型，用于为 Node.js 包的函数调用序列中的每一个 token 映射一个能保留单词语义信息的词向量。该词嵌入模型可以作如下描述：

$$TV_p \leftarrow E_W(S_p) \quad (3.1)$$

其中E_W为词嵌入方法，该方法将所有经过预处理的 Node.js 包的函数调用调用序列S_P作为训练语料，通过训练最终学习到一个映射函数TV_P:

$$TV_P: W_P \rightarrow V_{PW} \quad (3.2)$$

其中W_P是从函数调用序列文本S_P中获得的所有词汇，V_{PW}是词汇表W_P中所有词汇对应的向量空间。

将单词转换为词向量最简单的方法是使用单词的独热码，如果E_W采用独热码的编码方法，最大的弊端在于词向量不包含语义信息，这就可能会使得包含相似函数名称的函数调用序列可能会得到不同的嵌入结果，这不是本文方法所期望得到的。因此本文使用经典的词嵌入方法 word2vec 作为E_W，在词嵌入模型的训练过程中，上下文的窗口大小被设置为 4，词向量的维度设置为 100，最终学习到的映射函数TV_P可以将函数调用序列中的每一个 token 映射成一个能够保留单词语义信息的 100 维的数值向量。

3.2.2 卷积模型设计

得到词向量之后，还需要进一步将函数调用序列进一步转换成卷积神经网络能够处理的灰度矩阵的输入形式。对于 Node.js 包的函数调用序列$S = (t1, t2, t3, \dots, tn)$，假设 V_s是 S 转换之后的二维数值向量，那么关于 V_s 的公式成立：

$$V_s = V_{pw}(t1) \oplus V_{pw}(t2) \oplus V_{pw}(t3) \oplus ... \oplus V_{pw}(tn))$$.....(3.3)

其中⊕是连接运算符，即按照 token 在函数调用中的顺序从上之下依次将 token 的词向量进行拼接，完成S_P到V_s的转换。

由于卷积神经模型的输入应该是一个固定大小的二维数值向量，而 Node.js 包的函数调用序列长度却不是固定的。为了统一卷积神经网络模型的输入大小，本文统计了所有函数调用序列的 token 个数，并将函数调用序列的最大长度阈值设置为 L。对于长度大于 L 的函数调用序列，在将它们转换为二维向量之前去除多余的部分。对于长度小于 L 的函数调用序列，在转换是会附加零向量。通过这种处理方式，确保模型的输入大小保持一致，为 L*D（其中 D 为 token 词向量的维度 100）。那么对于软件包 apidoc-ui 的函数调用序列而言，在转换成卷积模型可以处理的输入向量之后，输入向量的第一行为单词 join 对应的词向量，最后一行为单词 fs 对应的词向量。

将 Node.js 包的函数调用序列转化为卷积神经网络模型可以处理的二维向量后，需要学习另一个可以将输入的二维向量映射成高维数值向量的映射函数PV，该数值向量即为 Node.js 软件包的特征向量，这个阶段称为 Node.js 包的嵌入阶段。映射函数PV可以由如下公式定义：

$$PV \leftarrow E_p(VS)$$.....(3.4)

其中 E_p为 NTR 中的卷积神经网络模型，VS为由函数调用序列集合转换而成的二维数值向量集合，需要注意的是VS是一组V_s（ $V_s \in VS$）。卷积神经网络训练结束后可以学习到映射函数PV。映射函数PV的定义如下：

$$PV: VS \rightarrow VSN$$.....(3.5)

其中VSN为 Node.js 包的特征向量集合。

NTR 中卷积神经网络模型是一个包含了两层卷积层，两层池化层和一个多层感知器的神经网络结构。网络的训练过程描述如下：第 1 层卷积层具有 20 个不同的卷积核，卷积核的大小设置为（2，D），需要注意的是使用卷积神经网络处理文本序列时，必须确保第 1 层卷积层中卷积核的维度大小和 token 词向量维度大小相等，同时步长设置为 1。输入 L*D 的二维向量经过第 1 层卷积之后就可以得到 20 个不同的特征图。第 2 层是最大池化层，池化窗口的大小设置为（2，1），步长设置为 2，第 1 层输出的 20 个特征图经过最大池化之后，特征图数量

不变，维度减半。第 3 层卷积层对经过最大池化后的 20 个特征图进行卷积，该卷积层中包含 50 个不同的卷积核，卷积核大小设置为（2，1），步长设置为 1，卷积之后得到 50 个新的特征图。这 50 个特征图经过第 4 层最大池化，池化窗口大小和步长依旧分别 s 设置为（2，1）和 2，得到 50 个新的维度减半的特征图。至此，卷积神经网络完成了函数调用序列局部特征的抽取。模型的后三层全连接，共同组成了一个多层感知器的结构用于函数调用序列局部特征的组合和压缩。多层感知器的输入模型第 4 层池化后的 50 个特征图，这 50 个特征图被拉长（flatten）全连接到包含 1000 个神经元的隐藏层，隐藏层的输出特征即为模型学习到的 Node.js 软件包的代码嵌入。隐藏层同样全连接到输出层，输出层维度大小设置为输入向量拉长后的维度大小。NTR 将 Node.js 软件包代码嵌入的学习任务被当作回归任务处理，因此输出层神经元的激活函数使用 softmax 函数。模型同样使用了平均绝对误差（Mean Absolute Error）作为损失函数，使用梯度下降算法寻找最优解。但是和监督学习不一样，NTR 中卷积神经网络模型是没有目标输出的，它只是单纯用来捕获函数调用序列的文本特征，并将其转换为数值向量而非完成某一项具体的分类任务。

综上，整个 Node.js 软件包特征向量的学习过程可以描述为映射函数PE的学习过程，映射函数PE的描述如下：

$$PE：S_p \rightarrow VSN \quad \text{.....(3.6)}$$

PE函数实现了 Node.js 包的函数调用序列到数值向量的转换，后续 NTR 将使用包的特征向量进行标签推荐。

3.3 多层感知器多标签分类

NTR 方法通过多标签分类来实现 Node.js 软件包的标签推荐。现针对 NTR 标签推荐过程作出如下形式化描述：所有的 Node.js 软件包共同组成了一个包集合$P = \{p_1, p_2, \dots, p_n\}$, 其中每一个$p_i$(1≤i≤n)代表一个 Node.js 软件包。统计软件包标签的词频，去除词频低于阈值的标签之后，剩下的标签组成了一个标签集合$\Sigma = \{t_1, t_2, \dots, t_w\}$。通过聚类算法，标签集合中的所有标签会被划分到不同的标签集群下，从而形成一个标签集群集合$\Sigma_{TC} = \{t_{c1}, t_{c2}, \dots, t_{ck}\}$，其中每一个标签集群$t_{ci}$都包含多个具有相似语义信息的标签。假设 Node.js 包p_i具有一个标签序

列$\{t_i, \ldots, t_j\}$，根据标签集群集合Σ_{TC}，p_i可以对应到一个标签集群序列$\{t_{cl}, \ldots, t_{cm}\}$。对于多层感知器的训练，包$p_i$对应一个输入输出对（$x_{Pi}$, y_{Pi}），其中 x_{Pi}为p_i的向量表示（$x_{pi} \in VSN$），y_{Pi} 为输入样本的类别向量，其维度大小为聚类之后标签集群的个数，每个维度均为 0 或者 1，其中为 1 代表p_i具有该维度对应标签集群中的标签，为 0 则相反。训练完成的网络模型可以为输入的 Node.js 包计算每一个标签集群的概率值，概率值靠前的 k 个标签集群中的标签将被作为候选的推荐标签。

表 3.3　标签推荐伪代码

标签推荐伪代码
输入：Node.js 软件包集合 $\boldsymbol{P} = \{\boldsymbol{p_1}, \boldsymbol{p_2}, \ldots, \boldsymbol{p_n}\}$
软件包的标签集合 $\boldsymbol{\Sigma} = \{\boldsymbol{t_1}, \boldsymbol{t_2}, \ldots, \boldsymbol{t_w}\}$
标签集群 $\boldsymbol{\Sigma_{TC}} = \{\boldsymbol{t_{c1}}, \boldsymbol{t_{c2}}, \ldots, \boldsymbol{t_{ck}}\}$ = K-means($\boldsymbol{\Sigma}$)
分类结果$\check{\boldsymbol{Y}}$= MLP($\boldsymbol{P}$, $\boldsymbol{\Sigma_{TC}}$)
输出：$\check{\boldsymbol{Y}}$中概率值最高的 k 个标签集群
Function MLP（$\boldsymbol{P}$， $\boldsymbol{\Sigma_{TC}}$）
输入向量 X = $\mathbf{PE}(\mathbf{P})$
输出向量 Y = $\boldsymbol{\Sigma_{TC}}$样本类别向量
训练集$\boldsymbol{\mathcal{D}} = \{(\boldsymbol{X}, \boldsymbol{Y})\}$
REPEAT
FOR ALL $(\boldsymbol{x^i}, \boldsymbol{y^i}) \in \boldsymbol{\mathcal{D}}$ DO
向前传播，计算出当前样本的输出$\check{\boldsymbol{y}}^{\boldsymbol{i}}$
计算预测输出和真实输出之间的误差
计算输出层神经元的梯度项$\boldsymbol{g_j}$
计算隐藏层神经元的梯度项$\boldsymbol{e_h}$
根据下降梯度计算层与层之间的权重和阈值
END
UNTIL 达到迭代次数
RETURN 权重和阈值确定的多层感知器模型 MLP

分类的第一步是对软件包的标签进行聚类。这样做既能对模型的输出进行统一化的管理，又能够增加推荐的标签数量。

标签聚类首先需要训练词嵌入模型获取每一个标签对应的词向量。由于 Node.js 软件包的标签是由开发人员人为定义的，可能会存在没有实际含义的软件标签。因此在执行标签聚类之前，需要统计标签出现的词频，删除低频的标签，以确保剩余的标签具有推荐价值。为了确保这些高频出现的标签都有词向量，即为了避免高频标签单词出现 OOV 问题，本文将包的标签序列信息和包的描述文本信息拼接到了一起，共同组成词嵌入模型的训练语料。在词嵌入模型训练时，词向量维度大小设置为 300，最小词频设置为 1，窗口大小设置为 4，学习率大小设置为 1e-2。词嵌入模型训练结束后，可以得到每一个高频出现标签的词向量。在聚类的时候，本文选择的是 K-means 聚类算法，并结合了手肘法和轮廓系数法来确定 K 的最优值。其中，手肘法通过计算误差平方和 SSE（Sum of the Square Errors）来决定最优 K，而轮廓系数法是通过计算平局轮廓系数 SC（Silhouette Coefficient）来决定最优 K。最终，所有高频出现的标签将被划分到了 K 个不同的标签集群下，与此同时，每一个 Node.js 软件包又可以根据自身的标签被分配到不同的标签集群当中，从而得到输入样本的类别向量。

分类的第二步就是训练多标签分类模型进行标签推荐。

在 NTR 中，标签推荐工作被当作成了多标签分类任务来处理，多标签分类就是将输入样本划分到标签集合中的多个标签下。常见的多标签分类方法包括二元关联，分类器链，ML-kNN[20]，ML-DT[35]以及神经网络。其中，前四种分类方法适用于判断输入样本是否属于某一个类别，但是 NTR 目的是为输入样本推荐 k 个最相关的标签而非直接分类。和其他多标签分类方法相比，深度神经网络可以为输入样本计算出每一个标签集群的概率，只需要选择出预测结果中概率值靠前的 k 个集群就可以实现标签推荐的目的，更加适用于标签推荐的场景。因此 NTR 选择使用深度神经网络作为多标签分类模型进行标签。具体地，本文搭建了一个多层感知器网络结构，多层感知器是由输入层、隐藏层以及输出层组成，每层之间互相全连接。输入层输入的是 Node.js 包的特征向量VSN，其维度等于 Node.js 包特征向量的维度。在隐藏层中，每个神经元的激活函数采用主流的 ReLU 激活函数，输出层维度的大小等于标签集群的个数，输出层神经元的激活

函数必须使用 sigmoid 激活函数，它可以预测每个输出维度为 1 的概率。模型的训练是通过不断降低预测值和真实值之间的二元交叉熵来完成的，并且结合梯度下降算法寻找神经元参数的最优值。通过设置最大迭代次数，判断模型是否收敛。最终可以学习到 Node.js 软件包代码向量到标签集群的映射函数 £，其定义如下：

$$£: R^{T} \rightarrow TC^{k@K} \quad \text{.....(3.7)}$$

其中R^{T}表示维度为 T 的 Node.js 包特征向量，$TC^{k@K}$表示 Σ_{TC} 中概率值靠前的 k 个标签集群。图 3.5 描述了 NTR 标签推荐的过程。

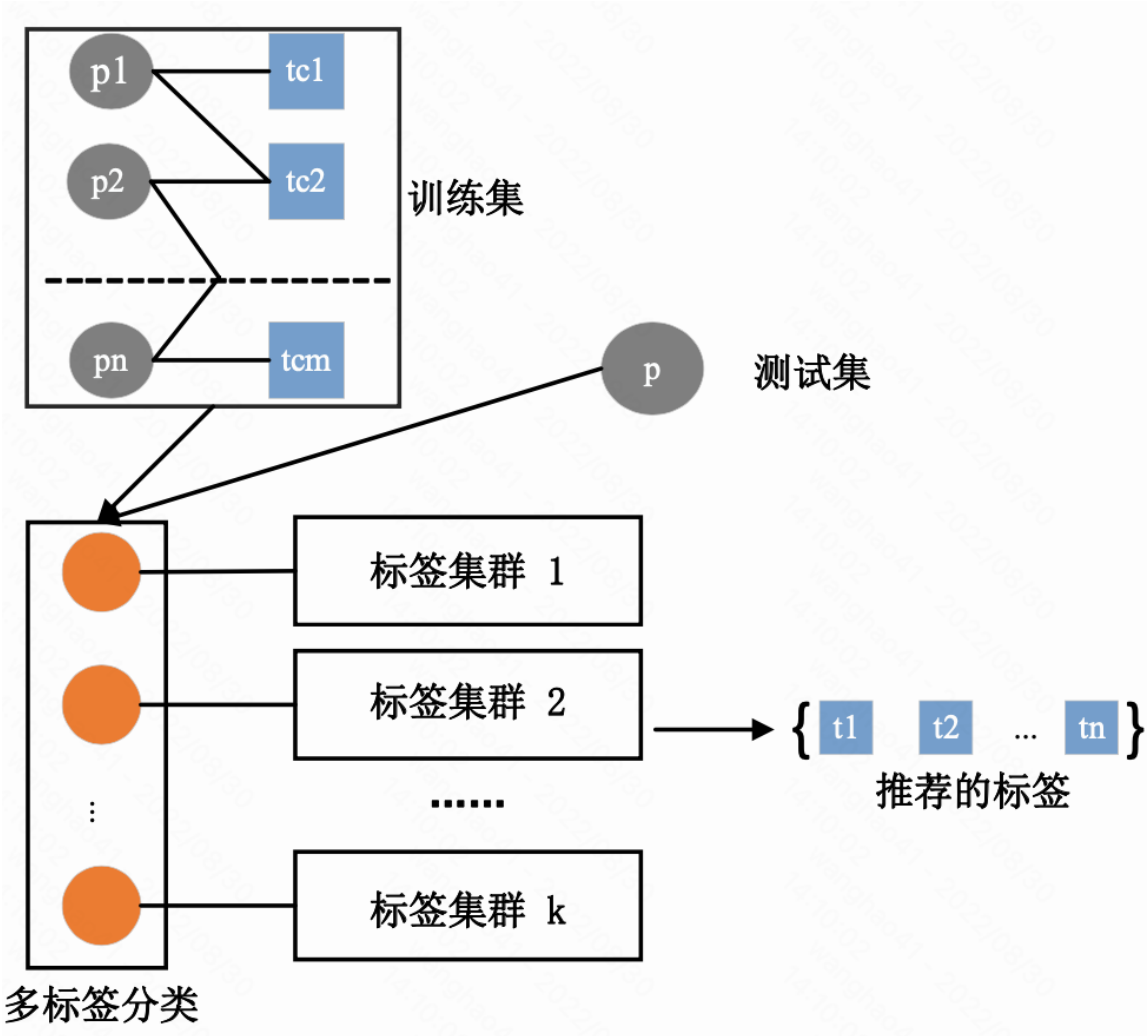


图 3.5　标签推荐模型的结构

对于软件包 apidoc-ui 而言，真实标签序列为(api，apidoc，document，restful，rest，doc)。表 3.4 中列出了 NTR 方法为 apidoc-ui 推荐的概率值靠前的 3 个标签集群，由表可见集群 top1 和 top3 推荐的标签和接口风格相关，集群 top2 推荐的标签和文档相关，整体的标签推荐结果和真实的标签结果是接近的。

表 3.4　NTR 推荐结果展示

推荐标签集群	标签结果
推荐标签集群 top1	crud　restful　rest
推荐标签集群 top2	markdown　apidoc　guide　comment　document　typedoc　doc　jsdoc
推荐标签集群 top3	server client handler api request middleware service

3.4 本章小结

综上所述，本章总共分三个小节介绍了 NTR 的具体实现，首先是利用 ECMAScript 和 Madge 工具对 Node.js 软件包的 JavaScript 代码进行静态程序分析，构建出 Node.js 软件包的函数调用图；其次是利用卷积神经网络结合词嵌入技术将 Node.js 包的函数调用序列转换成数值向量，即完成了 Node.js 包的嵌入工作；最后是训练多标签分类模型，将 Node.js 包的特征向量分类到了不同的标签集群中，实现了 Node.js 包的标签推荐。后续的章节将设计实验验证 NTR 的有效性。

第 4 章 实验设计与评估

NTR 是一种基于代码嵌入的标签推荐方法。除了需要评估 NTR 标签推荐的性能表现之外，本文还希望探究 NTR 学习到的 Node.js 包代码嵌入结果的好坏。因此综合考虑，实验环节提出并解决了如下的两个研究问题：

1) 研究问题Ⅰ：本文提出的嵌入方法是否可以准确学习到 Node.js 软件包源代码的向量化表示？
2) 研究问题Ⅱ：本文提出的标签推荐方法 NTR 是否可以合理地为 Node.js 软件包推荐标签？

4.1 数据收集及参数设置

为了能够收集到足够数量的 Node.js 软件包，本文首先使用 NPM 官网上推荐的关键字作为索引值编写爬虫程序抓取每一个关键字下所有 Node.js 软件包的名称等信息。需要声明的是，同一个关键字下的 Node.js 包可以被视为同一个类别。获取到 Node.js 包的名称信息之后，紧接着使用 NPM 的命令行指令 “npm install 包的名称”，将 Node.js 包下载到本地计算机用于后续的程序分析，最终形成了包含 34,977 个 Node.js 软件包的数据集。对于第一个研究问题，本文参考了 Import2vec[15]关于 Node.js 软件包向量可视化实验部分的数据，从本文的数据集中选择了 4 类关键字，每一个关键字下分别随机挑选了 1000 个包，总计 4000 个包用于向量的可视化分析。这四个类别的关键词分别是“grunt”，“gulp”，“homebridge”以及“react”。对于第二个研究问题，需要收集标记完好的 Node.js 包用于多标签分类模型的监督训练，为此手工挑选了标记完好且下载量较高的 Node.js 软件包，去除了词频低于 5 的标签之后，最终形成了包含 3,831 个软件包，695 个有效标签的数据集。除去上述俩实验中 7,318 个包数据，包数据集中剩下的 27,596 个 Node.js 包用于嵌入模型的训练。

NTR 中卷积神经网络的输入是一个固定大小的 L*D 的二维数值向量矩阵，其中 L 是 Node.js 软件包的函数调用序列长度，D 是调用序列中词向量的维度。由于包的函数调用序列长度不一致，为了确定一个统一且合理的最大长度阈值 L，本文从 34,977 个 Node.js 包中提取出函数调用序列，并且统计了所有函数调用序列的长度，统计结果表明大部分 Node.js 包的函数调用序列长度介于 20 到 100 之间，但是 L 设置为 100 只能覆盖到 75.1%的 Node.js 软件包。因此为了保留更多

Node.js 软件包函数调用序列的完整信息，L 被设置为 300，覆盖率达 97.1%。对于词向量维度 D，被设置为经典值 100，因此卷积神经网络的输入大小固定为 300*100。

表 4.1 重要模型参数设置

word2vec 模型的参数设置	
学习率	1e-2
最小词频	1
窗口大小	4
词向量维度	100
卷积神经网络参数设置	
学习率	1e-2
损失函数	平均绝对误差
优化算法	随机梯度下降
池化类型	最大池化
隐藏层神经元数	1000
输出层激活函数	softmax
其他层激活函数	ReLU
多层感知器参数设置	
损失函数	二元交叉熵
隐藏层神经元数	500
输出层激活函数	sigmoid
隐藏层激活函数	ReLU

表 4.1 分别列出了 NTR 中词嵌入模型，卷积神经网络模型以及多标签分类模型的其他参数设置。其中，在设置词嵌入模型 word2vec 以及卷积神经网络其他参数时，本文综合考虑了论文[16,34]中具体实现选择出了适合本文场景的网络参数。对于执行多标签分类任务的多层感知器而言，由于本文处理的数据相对简单，隐藏层数为 1 到 2 层就可以，同时按照隐藏层神经元的个数大小应该为输入层大小 2/3 加输出层大小 2/3 的标准设置为 500；对于隐藏层激活函数的设置，本文选择了 ReLU 作为激活函数，因为与 sigmoid 和 tanh 激活函数相比，ReLU 可以

加速模型的收敛；在输出层，必须使用 sigmoid 作为输出层神经元的激活函数，用于每个标签集群概率分数的计算；模型使用二元交叉熵作为损失函数和并结合随机梯度下降优化算法进行训练。

4.2 Node.js 包嵌入可视化（研究问题Ⅰ）

在实体嵌入的工作中，往往会对学习到的高维特征向量进行降维可视化，然后通过向量在低维空间内的位置关系去评估嵌入方法的有效性。因此针对第一个研究问题，本文打算从 Node.js 包向量之间相邻性的角度进行分析。相邻性是指在同一关键词下的 Node.js 包在数据降维之后是否在向量空间中更相邻。为此，本文采用了 t-SNE[31]技术对 1000 维的 Node.js 包向量进行数据降维，然后使 Matplotlib 库和 TensorBoard 对降维后的结果进行可视化分析。本文提出了如下的两个假设：原始假设（H0）：不同关键词下 Node.js 软件包的向量降维可视化结果没有显著差异。备择假设（H1）：不同关键词下 Node.js 软件包的向量降维可视化结果有显著差异。那么如果向量降低维度之后，相同关键字下的 Node.js 软件包分布相对集中，则可以认为假设 H1 成立。

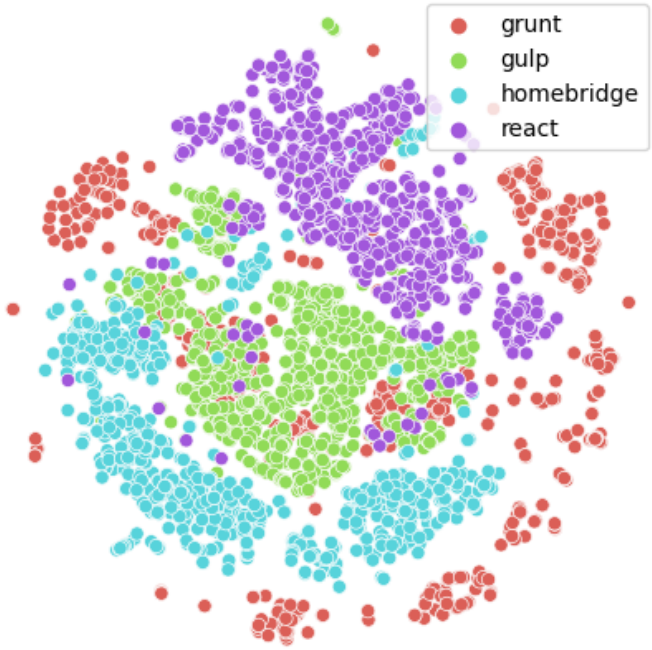


图 4.1　降维可视化结果

关于相邻性，图 4.1 描绘了这 4,000 个 Node.js 包向量通过 t-SNE 进行数据降维之后的二维向量空间。其中类别 0-3 依次代表了“grunt”,“gulp”,“homebridge”以及“react”这四类关键词。从图中可以直观地看出，尽管会存在一些离散的点，

但相同关键字下的 Node.js 软件包在向量空间中的分布相对集中，假设 H1 成立。有一个需要被考虑的情况是红色点代表的‘grunt’关键词下的包没有全部集中分布在一起而是被拆分多个不同的小簇，这是因为同一个关键字下的包也有可能实现不同的功能，从而对应着不同的函数调用序列，导致图中出现同一个关键字下的包被分成几个簇的现象，比如 Grunt 本身是一个前端自动化构建工具，在‘grunt’关键词下可能会存在处理不同类型文件的插件。

因此从相邻性的角度考虑，可以得出结论：本文提出的 Node.js 包代码的嵌入方法是可以准确学习到 Node.js 软件包源代码的向量化表示的。

4.3 NTR 标签推荐结果（研究问题Ⅱ）

针对第二个研究问题“本文提出的标签推荐方法 NTR 是否可以合理地为 Node.js 包推荐标签？”，本小节将围绕以下三个方面展开：标签聚类，多标签分类模型性能的对比以及 NTR 在标签推荐上面的表现。

4.3.1 标签聚类

NTR 通过分类模型将 Node.js 包分到不同类别的标签集群下，然后选择概率值靠前的标签集群内的标签作为推荐结果，因此标签集群的数量决定了模型的输出维度，从而直接影响了模型的性能表现。如何对标签数据进行聚类以及将标签聚到多少个集群下成为一个很重要的问题。在该步骤中本文使用了 K-means 算法将所有 Node.js 包的标签按照标签的词向量划分到了不同的标签集群下，以便于后续的分类任务。具体步骤如下：首先，本文统计了数据集中标签的词频并且过滤掉了出现频率低于 5 的标签，同时手动筛选出了没有实际含义的标签，最终得到 595 个标签数据；紧接着，训练了一个词嵌入模型用于生成标签的词向量。为了避免 OOV 问题，确保所有的高频标签都具有对应的词向量，需要构建一个由 Node.js 包的描述信息和标签信息组合而成的语料库用于词嵌入模型的训练；最后使用 K-means 聚类算法对标签的词向量进行聚类，对于 K 最优值的确定，本文结合了手肘法和轮廓系数法作为理论依据，将 K 值设置为 50 和 100, 也就是分别将 695 个标签根据词向量的语义信息聚到了 50 个标签集群和 100 个标签集群下。表 4.2 中展示了 K 值为 50 时部分的标签聚类结果。从表中的数据可以看出，聚类结果是合理的。

由于 K 值的大小直接决定了模型的输出维度，因此随着 K 值的增加，模型

的预测难度会增大，分类表现势必会变差。在后续的实验中，本文将会考虑不同 K 值对模型分类结果影响的具体表现。

表 4.2　K=50 标签聚类结果展示

标签集群	标签
1	dev, install, git, npm, coverage, github, issue, coveralls, license, package, release, report, changelog, dependency,
2	websql, mongodb, postgresql, backend, mongo, query, data, postgres, storage, secure, sqlite, mysql, database, mssql, oracle,
3	socket, transport, connect, protocol, http2, openidconnect, client, websocket, proxy, http,

4.3.2 分类模型对比

NTR 借助于多标签分类模型的分类结果进行标签推荐。而在多标签分类模型的选择上，NTR 选择了深度神经网络作为分类模型，具体是一个隐藏层数为 2 的多层感知器。本小节主要用于验证和其他的多标签分类模型相比，深度神经网络模的分类结果是否更好？

下面将介绍在多标签分类场景下的评估指标[36]：汉明损失 Hamming Loss，精确率 Precision，召回率 Recall 以及 F1 值。汉明损失是衡量在整个样本中，预测错误的标签个数的占比，因此该值越低则说明模型的分类能力越强；精确率，召回率以及 F1 值都是用来计算样本的平均精确率，其中精确率反应了在整个样本中预测正确标签在预测结果中的比例，而召回率则反应了在整个样本中预测正确的标签在样本真实标签上的比例。具体的定义如下：

- Hamming Loss：

$$Hamming\ Loss(y^t, y^p) = \frac{1}{|M|}\sum_{i=1}^{|M|}\frac{xor(y^t, y^p)}{|N|} \quad \text{.....(4.1)}$$

- Precision:

$$Precision = \frac{1}{|M|}\sum_{i=1}^{|M|}\frac{|y^t \cap y^p|}{|y^p|} \quad \text{.....(4.2)}$$

- Recall:

$$Recall = \frac{1}{|M|}\sum_{i=1}^{|M|}\frac{|y^t \cap y^p|}{|y^t|} \quad \text{.....(4.3)}$$

■ F1 Score:

$$F1 = \frac{1}{|M|}\sum_{i=1}^{|M|}\frac{2|y^t \cap y^p|}{|y^t|+|y^p|} \quad (4.4)$$

其中M代表样本集合，N代表标签集合，y^t和y^p分别代表样本真实的类别向量和预测的类别向量。

本文对比了 5 种多标签分类模型，分别为方法论提到的二元关联 BR，分类器链 CC，多标签决策树 ML-DT，多标签 k 近邻 ML-kNN 以及深度神经网络 MLP。对于所有的分类模型，实验数据均为 3833 个 Node.js 软件包和对应的 595 个软件标签。按照传统的二八划分，将实验数据划分为不相交的训练集和测试集，分别考虑将表标签聚到 50 个不同的集群和 100 个不同的集群。表 4.3 分别列出了这 5 种模型在 K=50 和 K=100 时性能表现。其中，对于多标签 k 近邻算法，k 值设置在 5 到 10 之间可以取得最佳的实验结果，对于多层感知器的神经网络结构，模型迭代 200 轮左右就可以取得如下效果。图 4.2 为多层感知器损失函数的下降图。

表 4. 3　多标签分类模型性能对比

	K=50				K=100			
	HL	P	R	F1	HL	P	R	F1
BR	0.296	0.202	**0.661**	0.288	0.259	0.137	**0.582**	0.210
CC	0.243	0.224	0.648	0.311	0.152	0.167	0.566	0.239
ML-DT	0.067	0.690	0.470	0.525	0.049	0.561	0.522	0.514
ML-kNN	0.065	0.714	0.561	0.597	0.035	0.684	0.531	0.568
MLP	**0.064**	**0.723**	0.608	**0.628**	**0.033**	**0.721**	0.572	**0.604**

表 4.3 中的数据能直观的反映出和其他多标签分类方法相比，多标签 k 近邻算法和深度神经网络的分类效果更佳。但是对于二元关联方法和分类器链方法，它们却有异常的高的召回率，原因在于这两种方法是通过训练 n 个二分类器进行多标签分类，这就导致预测结果中会有更多的标记，而 R 值则是反应预测结果中正确预测的标签在真实结果中的比例，预测的标签越多，在真实结果上的比例就会越高，同时错误预测的标签个数也会增加，这就导致了 R 值很高，其他指标却很糟糕的情况。此外，另有一个需要解释的现象是汉明损失会随着 K 值的增大而减小，这是因为汉明损失反映了错误标签在标签集上的占比，当 K 值增加时，

标签集个数增加，相当于分母增加从而会导致错误标签的比例降低。

实验结果也证明了 NTR 方法使用多层感知器作为多标签分类模型进行标签推荐是合理的。

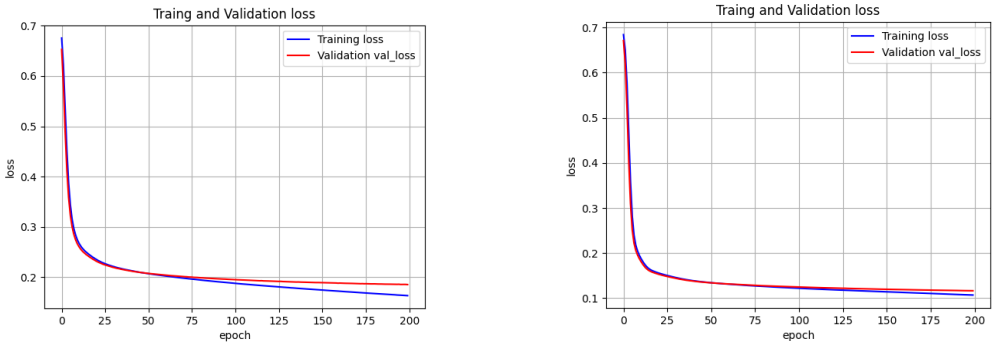


图 4. 2　K=50 和 K=100 时训练集和验证集损失函数下降图

4.3.3 标签推荐表现

在标签推荐表现的评估上面，本文与其他标签推荐工作[2,4,5]的评估指标保持一致，采用了 Precision@k，Recall@k 和 F1-score@k 来评估 NTR 的标签推荐性能。其中，Precision@k 则是反映正确推荐的标签集群在 k 个推荐的标签集群所占的比率，Recall@k 反映正确推荐的标签集群在 Node.js 包的真实标签集群中所占的比率，因此 Recall@k 往往远大于 Precision@k，F1-score@k 的值则是根据 Recall@k 和 Precision@k 计算得到的，他们的具体定义如下：

- Precision@k:

$$Precision@k = \frac{1}{n}\sum_{i=1}^{n}\frac{\left|TagClusters_i^{topk} \cap TagCluster_i^{correct}\right|}{\left|TagClusters_i^{topk}\right|} \quad \text{.....(4.5)}$$

- Recall@k:

$$Recall@k = \frac{1}{n}\sum_{i=1}^{n}\frac{\left|TagClusters_i^{topk} \cap TagCluster_i^{correct}\right|}{\left|TagClusters_i^{correct}\right|} \quad \text{.....(4.6)}$$

- F1-score@k:

$$F1-\text{score@k} = 2 * \frac{Precision@k * Reacll@k}{Precision@k + Reacll@k} \quad \text{.....(4.7)}$$

其中 $TagClusters_i^{topk}$ 表示分类模型计算出概率值前 k 的标签集群，$TagCluster_i^{correct}$ 表示 Node.js 包所属的真实的标签集群。

针对该实验，本文同样使用 3833 个 Node.js 软件包和 595 个高频标签作为实验数据，并且同时考虑了将标签聚到 50 个不同的集群和 100 个不同的集群下

这两种不同的情况，实验数据按照传统的二八划分形成了不相交的测试集和训练集。NTR 通过分类模型对 Node.js 包的代码嵌入多分类进行标签推荐，代码嵌入模型的训练语料为 Node.js 包的函数调用序列。基于 NTR 的具体实现，本文衍生出了另外两种子方法 NTR_PN 和 NTR_MN。NTR_PN 是在生成 Node.js 软件包的函数调用序列时，添加包的名称信息；NTR_MN 则是在生成 Node.js 软件包的函数调用序列时，如果一个函数调用是 Node.js 内置模块提供的 API，在函数调用序列中添加该函数名称的同时加上内置模块的名称信息。由于这两种文本信息对于 Node.js 包而言至关重要，因此本文通过在函数调用序列中添加这两种信息进行代码嵌入，从而衍生出了 NTR_PN 和 NTR_MN 两种方法。

除了需要探究 NTR 方法本身在标签推荐上的性能表现之外，本文还希望探究和其他的标签推荐方法相比，NTR 是否真的更适用于这种描述文本不充足甚至缺失的标签推荐场景？需要增加对比实验验证 NTR 方法的不可替代性。NTR 方法的最大特点是利用的 Node.js 软件包的代码文本推荐标签，是一种基于代码嵌入的标签推荐方法，而其他大多数基本文本的标签推荐方法都是利用软件项目的描述文本。因此在对比实验的设置中，需要抽取 Node.js 软件包的描述文本特征用于对比方法的训练。在对比方法的选择上，本文选择了和传统的标签推荐方法 EnTagRec[5]和基于深度学习的标签推荐方法 tagCNN[6]作对比。EnTagRec 是一种利用软件项目的描述文本以及项目开发者的历史标签信息进行标签推荐。由于 Node.js 软件包作者的历史标签信息难以获取，因此在该对比实验中仅仅使用了 Node.js 软件包的描述文本信息作为 EnTagRec 的数据源。tagCNN 则是通过训练卷积神经网络对软件包的描述文本进行回归预测，这种方法和 NTR 的思路相似，但区别在于 NTR 是一种基于代码嵌入的标签推荐方法，和使用描述文本进行分类的方法相比，更加适用于 Node.js 软件包的标签推荐场景。

表 4.4 列出了具体的对比实验结果，由表中的数据可以看出 NTR 及其衍生的标签推荐方法 NTR_PN 和 NTR_MN 在 K 为 50 和 K 为 100 时，三项指标的性能表现均优于利用描述文本进行标签推荐的方法 tagCNN 和 EnTagRec。而 tagCNN 和 EnTagRec 在三项指标上的表现相差则不大。这是因为简短且残缺的描述文本很难准确概括一个包的特征，而 Node.js 软件包的源代码却可以直接反应出一个包的功能特性。因此在 Node.js 软件包的标签推荐场景下，基于代码嵌

入的标签推荐方法 NTR 可以比利用描述文本进行标签推荐的方法 tagCNN 和 EnTagRec 取得更高的 Recall@k 值，Precision@k 值以及 F1-score@k 值。在 NTR，NTR_PN 和 NTR_MN 的对比中，实验结果证明在函数调用序列中添加额外的文本信息不会使标签推荐的结果更优，反而会降低推荐结果的准确率。导致这种现象的原因可能是因为在函数调用序列中添加额外的文本信息后，Node.js 软件包的代码嵌入包含的特征相会对增多，从而加大了模型的分类难度，使得标签推荐结果准确率下降。

针对具体的实验结果，本文还分析了实验中存在有效性威胁和潜在风险。一个是由于 Node.js 包的标签是人为定义的，不同开发者管理的功能相似的 Node.js 包可能拥有一些开发者个人特色的标签。在这种情况下，相似的输入会映射到不同的输出上，从而增加了模型预测的难度。另一个就是在标签聚类的过程中，聚类算法并不能确保在同一个标签集群下的标签语义信息完全相似，而是只能尽量去满足这个条件，因此推荐的标签集群中难免会有噪声标签。

表 4.4　标签推荐实验结果

K=50						
	result@5			result@10		
	P	R	F	P	R	F
tagCNN	0.436	0.578	0.497	0.305	0.729	0.430
EnTagRec	0.433	0.520	-	0.306	0.670	-
NTR	**0.489**	**0.640**	**0.554**	**0.329**	**0.768**	**0.460**
NTR_PN	0.482	0.637	0.549	0.324	0.763	0.455
NTR_MN	0.478	0.626	0.542	0.323	0.756	0.452
K=100						
tagCNN	0.394	0.530	0.452	0.267	0.646	0.378
EnTagRec	0.416	0.500	-	0.308	0.673	-
NTR	**0.464**	0.601	0.523	**0.309**	0.713	**0.431**
NTR_PN	0.463	**0.607**	**0.525**	0.308	**0.715**	0.430
NTR_MN	0.454	0.593	0.514	0.305	0.699	0.424

4.4 本章小结

实验环节部分主要介绍了数据收集过程和模型的参数设置，并且验证了俩个研究问题。实验结果表明本文提出的代码嵌入方法可以准确学习到 Node.js 包源代码的向量化表示，同时证实了在 Node.js 软件包标签推荐场景中，基于代码嵌入的标签推荐方法 NTR 的表现优于基于描述文本特征进行标签推荐的方法 EnTagRec 和 tagCNN。

第 5 章 总结与展望

5.1 总结

现有的标签推荐方法大多利用实体的文本特征作为数据源训练分类模型来实现。但在 Node.js 软件包的标签推荐场景下，Readme 文本噪声信息过多且预处理过程繁琐。包的描述文本过于简短甚至缺失，包含的有效信息太少。因此这两种文本特征都不适合作为 Node.js 包标签推荐模型的数据源。

考虑到描述文本和 Readme 文档的局限性，本文从 Node.js 软件包的代码文本出发，提出了一种基于代码嵌入的 Node.js 包标签推荐方法 NTR。NTR 主要由 Node.js 软件包的嵌入模块和多标签分类模块组成。Node.js 软件包的嵌入模块旨在训练嵌入模型学习软件包的代码向量用作分类模型的输入。在该模块下，NTR 首先利用了静态程序分析的技术解析包的 JavaScript 源代码，通过制定语法规则定位抽象语法树中函数节点构造出包的函数调用图。然后利用文本卷积神经网络作为包的嵌入模型，同时结合词嵌入技术将 Node.js 包的函数调用序列转换为数值向量，完成 Node.js 包的嵌入。多标签分类模块用于软件包的标签推荐。在该模块下，NTR 首先利用聚类算法将软件标签划分到了不同的标签集群下，然后搭建了一个多层感知器的神经网络模型，该模型以 Node.js 包向量为输入并在输出层计算不同标签集群的概率，选择概率值靠前的 k 个标签集群作为 NTR 方法的推荐结果。

在实验设计部分，本文验证了两个不同的研究问题“本文提出的嵌入方法是否可以准确学习到 Node.js 软件包源代码的向量化表示”和“本文提出的标签推荐方法 NTR 是否可以合理地为 Node.js 软件包推荐标签？”。实验结果证明：NTR 不仅可以学习到能够保留代码语义信息的 Node.js 包向量，并且在描述文本不充分甚至缺失的场景下标签推荐结果优于基于描述文本的标签推荐方法 EnTagRec 和 tagCNN。

5.2 展望

NTR 代码嵌入部分的原理是通过图遍历算法将 Node.js 软件包的函数调用图转换为可支持文本卷积神经网络处理的函数调用序列，并进行编码。但实际上，图作为一种由边集和点集构成的非线性数据结构，将其转换为函数调用序列的形

式可能无法充分描述图中结点之间边的关系，丢失图的部分信息，从而导致软件包编码结果不准确。故之后的研究可集中于对编码结构进行优化，使用图神经网络代替卷积神经网络来处理 Node.js 软件包的函数调用图[37]，使其能够极大地保留函数调用图的网络拓扑结构和节点属性信息，得到更加准确的调用图编码向量。

参考文献

[1] Al-Kofahi J M, Tamrawi A, Nguyen T T, et al. Fuzzy set approach for automatic tagging in evolving software[C]//2010 IEEE International Conference on Software Maintenance. IEEE, 2010: 1-10.

[2] Wang X Y, Xia X, Lo D. Tagcombine: Recommending tags to contents in software information sites[J]. Journal of Computer Science and Technology, 2015, 30(5): 1017-1035.

[3] Zhou P, Liu J, Yang Z, et al. Scalable tag recommendation for software information sites[C]//2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER). IEEE, 2017: 272-282.

[4] Wang S , Lo D , Vasilescu B , et al. EnTagRec: An Enhanced Tag Recommendation System for Software Information Sites[C]// International Conference on Software Maintenance and Evolution. IEEE, 2014.

[5] Wang S, Lo D, Vasilescu B, et al. EnTagRec++: An enhanced tag recommendation system for software information sites[J]. Empirical Software Engineering, 2018, 23(2): 800-832.

[6] Zhou P, Liu J, Liu X, et al. Is deep learning better than traditional approaches in tag recommendation for software information sites?[J]. Information and software technology, 2019, 109: 1-13.

[7] Lei K, Fu Q, Yang M, et al. Tag recommendation by text classification with attention-based capsule network[J]. Neurocomputing, 2020, 391: 65-73.

[8] Gu X, Zhang H, Kim S. Deep code search[C]//2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE). IEEE, 2018: 933-944.

[9] Peng H, Mou L, Li G, et al. Building program vector representations for deep learning[C]//International conference on knowledge science, engineering and management. Springer, Cham, 2015: 547-553.

[10] Mou L, Li G, Jin Z, et al. TBCNN: A tree-based convolutional neural network for programming language processing[J]. arXiv preprint arXiv:1409.5718, 2014.

[11] Alon U, Zilberstein M, Levy O, et al. A general path-based representation for

predicting program properties[J]. ACM SIGPLAN Notices, 2018, 53(4): 404-419.

[12] Alon U, Zilberstein M, Levy O, et al. code2vec: Learning distributed representations of code[J]. Proceedings of the ACM on Programming Languages, 2019, 3(POPL): 1-29.

[13] Zhang J, Wang X, Zhang H, et al. A novel neural source code representation based on abstract syntax tree[C]//2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE). IEEE, 2019: 783-794.

[14] Mateless R, Tsur O, Moskovitch R. Pkg2Vec: Hierarchical package embedding for code authorship attribution[J]. Future Generation Computer Systems, 2021, 116: 49-60.

[15] Theeten B, Vandeputte F, Van Cutsem T. Import2vec: Learning embeddings for software libraries[C]//2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR). IEEE, 2019: 18-28.

[16] Liu K, Kim D, Bissyandé T F, et al. Learning to spot and refactor inconsistent method names[C]//2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE). IEEE, 2019: 1-12.

[17] Nguyen S, Phan H, Le T, et al. Suggesting natural method names to check name consistencies[C]//Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering. 2020: 1372-1384.

[18] Malik R S, Patra J, Pradel M. NL2Type: inferring JavaScript function types from natural language information[C]//2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE). IEEE, 2019: 304-315.

[19] Tsoumakas G, Katakis I. Multi-label classification: An overview[J]. International Journal of Data Warehousing and Mining (IJDWM), 2007, 3(3): 1-13.

[20] Zhang M L, Zhou Z H. A k-nearest neighbor based algorithm for multi-label classification[C]//2005 IEEE international conference on granular computing. IEEE, 2005, 2: 718-721.

[21] Yang P, Sun X, Li W, et al. SGM: sequence generation model for multi-label classification[J]. arXiv preprint arXiv:1806.04822, 2018.

[22] Durand T, Mehrasa N, Mori G. Learning a deep convnet for multi-label classification with partial labels[C]//Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. 2019: 647-657.

[23] Mikolov T, Chen K, Corrado G, et al. Efficient estimation of word representations in vector space[J]. arXiv preprint arXiv:1301.3781, 2013.

[24] Bojanowski P, Grave E, Joulin A, et al. Enriching word vectors with subword information[J]. Transactions of the Association for Computational Linguistics, 2017, 5: 135-146.

[25] Karampatsis R M, Babii H, Robbes R, et al. Big code!= big vocabulary: Open-vocabulary models for source code[C]//2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE). IEEE, 2020: 1073-1085.

[26] Chen Z, Monperrus M. A literature study of embeddings on source code[J]. arXiv preprint arXiv:1904.03061, 2019.

[27] Sze V, Chen Y H, Yang T J, et al. Efficient processing of deep neural networks: A tutorial and survey[J]. Proceedings of the IEEE, 2017, 105(12): 2295-2329.

[28] LeCun Y, Bottou L, Bengio Y, et al. Gradient-based learning applied to document recognition[J]. Proceedings of the IEEE, 1998, 86(11): 2278-2324.

[29] Kim Y . Convolutional Neural Networks for Sentence Classification[J]. Eprint Arxiv, 2014.

[30] Pearson K. LIII. On lines and planes of closest fit to systems of points in space[J]. The London, Edinburgh, and Dublin philosophical magazine and journal of science, 1901, 2(11): 559-572.

[31] Van der Maaten L, Hinton G. Visualizing data using t-SNE[J]. Journal of machine learning research, 2008, 9(11).

[32] Hinton G E, Roweis S. Stochastic neighbor embedding[J]. Advances in neural information processing systems, 2002, 15.

[33] Xue S, Zhang L, Li A, et al. Appdna: App behavior profiling via graph-based

deep learning[C]//IEEE INFOCOM 2018-IEEE Conference on Computer Communications. IEEE, 2018: 1475-1483.

[34] Liu K, Kim D, Bissyandé T F, et al. Mining fix patterns for findbugs violations[J]. IEEE Transactions on Software Engineering, 2018, 47(1): 165-188.

[35] Chou S, Hsu C L. MMDT: a multi-valued and multi-labeled decision tree classifier for data mining[J]. Expert Systems with Applications, 2005, 28(4): 799-812.

[36] Sorower M S. A literature survey on algorithms for multi-label learning[J]. Oregon State University, Corvallis, 2010, 18: 1-25.

[37] Goyal P, Ferrara E. Graph embedding techniques, applications, and performance: A survey[J]. Knowledge-Based Systems, 2018, 151: 78-94.

作者在学期间学术成果

本人在学期间以除导师外第一作者身份撰写的论文《基于 seq2seq 模型的标签推荐方法》已被《吉林大学学报（理学版）》录用。

此外，本毕业论文的研究成果已投稿于软件工程领域 CCF 推荐 B 类期刊《Journal of Software: Evolution and Process》，正在外审中。

致谢

三年的时间交出了面前的这份毕业答卷，也为我的学生时代画上了句号。结局虽称不上完美，却是我努力过的最好证明。

2019 年本科毕业之后我选择继续留在本校读研，成为了刘磊老师和刘华虓老师底下的一名研究生。三年的时光很短暂，短暂到一切仿佛都发生在昨天。第一次给整个实验室的同学分享论文时的紧张感，读了很多论文却依旧没有研究方向时的迷茫以及多次试错之后模型最终成功顺利跑起来时的成就感，所有的场景依旧历历在目。三年的时光很漫长，漫长到足以让一个人完成专业素养和个人能力的蜕变。我所在的智能化软件开发团队是一个学术氛围浓厚的科研队伍，很庆幸能够成为其中一员。科研过程其实是一个团队合作的过程，每次遇到技术上的问题，组内的师兄和同学们会毫不吝啬地帮助我解答疑惑，积极分享出个人的见解。我的老师更是全程参与了我的科研工作，从最开始的选题到最终论文的修改，事无巨细尽职尽责。科研过程同样也是一个磨炼意志的过程，当接触到新的知识领域面对新的技术难题时，我会陷入自我怀疑觉得问题无解，或者觉得问题过于复杂而无从下手。其实当你开始花费精力和时间去尝试解决问题，无解的问题也会开始有眉目。面对技术难题时，要相信自己有解决问题的能力，克服自身的惰性，静下心来寻找解决问题的方案，这是我在科研过程中学到的最珍贵的道理，一生受用。感谢老师在这三年里的辛苦付出，感谢实验室成员的互相包容，也感谢科研过程中遇到的顺利和不顺，这三年将会是我人生中一段重要的时光。

求学十九载，我在父母和亲人的庇护下完成了我的学业。我妈经常用最朴素的语言给我讲述她的人生道理。如今，我即将离开校园，投身社会，希望自己能够谨记父母的叮嘱，不忘初心，早日有能力回馈家庭。