Repo2Vec:确定存储库相似度的综合嵌入方法

Md Omar Faruk Rokon
UC Riverside
mroko001@ucr.edu

裴岩
加州大学河滨分校
pyan012@ucr.edu

Risul Islam
UC Riverside
risla002@ucr.edu

Michalis Faloutsos
UC Riverside
michalis@cs.ucr.edu

***摘要*--**
我们如何在大型在线档案库（如GitHub）中识别相似的存储库和集群？确定软件库的相似性是研究这类软件生态系统的动态和演变的一个重要基石。关键的挑战是为不同的软件库特征确定正确的表示方法，以达到以下目的。(a)它可以捕捉到所有可用信息的各个方面，(b)它可以很容易地被ML算法使用。我们提出Repo2Vec，这是一种全面的嵌入方法，通过结合三类信息源的特征，将资源库表示为一个分布式矢量。作为我们的主要创新，我们考虑了三种类型的信息：（a）元数据，（b）资源库的结构，以及（c）源代码。我们还引入了一系列的嵌入方法来表示这些信息类型并将其结合到一个单一的嵌入中。我们用GitHub的两个真实数据集对我们的方法进行了评估，这些数据集共包括1013个仓库。首先，我们表明我们的方法在精确度方面优于以前的方法（93% vs 78%），强相似库的数量几乎是以前的两倍，误报率减少30%。其次，我们展示了Repo2Vec如何为以下工作提供坚实的基础。(a) 区分恶意软件和良性软件库，以及(b)确定有意义的分层聚类。例如，我们在区分恶意软件和良性软件库方面取得了98%的精确度和96%的召回率。总的来说，我们的工作是实现许多软件库分析功能的基本构件，如按目标平台或意图对软件库进行分类，检测代码重复使用和克隆，以及识别血统和进化。

***Index Terms*-Embedding, GitHub, Similarity, Clustering, Software.**

I. 简介

建立一种衡量软件库之间相似性的方法是研究在线开源软件（OSS）平台中大量软件库的重要基石。这些开放源码软件平台包含了大量的软件库和数百万用户的参与[1]。在这些平台上有大量的合作和代码重复使用[2], [3]，这些都是公开支持和鼓励的。研究人员对研究这类资源库的动态感兴趣，其中包括识别：（a）衍生资源库，（b）资源库家族，（c）软件项目的演变，以及（d）编码和技术趋势。GitHub可以说是最大的此类平台，拥有超过3200万个资源库和3400万用户，表现出显著的协作互动[4]。

我们怎样才能量化两个资源库之间的相似程度？这就是我们要解决的问题。以GitHub为例，每个资源库都由元数据、源代码和辅助文件组成。给定一个资源库，我们如何能

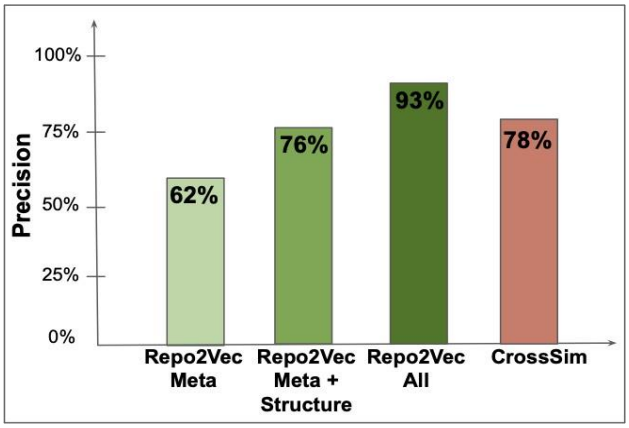

图1：我们的方法在使用CrossSim数据集的精确度方面优于最先进的方法CrossSim。我们还看到了Repo2Vec考虑的不同类型信息的效果：元数据、添加结构和添加源代码信息。

在一个大集合中确定最相似的存储库？这里的输入是大量的存储库和一组查询。希望的输出是：(a)对于一个给定的查询资源库，最相似的资源库；(b)相似资源库的集群。这里的关键挑战是将资源库数据表示为数字特征向量，以使ML方法能够计算出资源库之间的相似性和集群。此外，像我们在这里要做的那样，将不同类型的信息向量结合起来，也是一个挑战。

目前专注于建立资源库之间的相似性的工作相对较少，其中大多数使用元数据或源代码级别的信息，而没有一个使用我们这里的三类信息。首先，LibRec[5]、SimApp[6]、Collaborative Tagging[7]和RepoPal[8]只利用元数据来寻找存储库之间的相似性。第二，MUDABLUE[9]和CLAN[10]是两种只使用资源库的源代码作为纯文本的相似性计算方法。第三，CrossSim[11], [12]提出了一个图表示法，使用元数据和源代码计算资源库之间的相似性。我们在第七节中详细讨论相关的工作。

作为我们的主要贡献，我们提出了Repo2Vec，这是一种用多维分布式连续矢量表示软件库的方法，可以用来衡量库之间的相似性。我们

简要介绍一下我们方法的主要特点。首先，我们的方法将版本库表示为嵌入空间中的分布式连续矢量。其次，我们考虑三种类型的信息：（a）元数据，（b）源代码，和（c）资源库目录结构。我们的方法提供了一种灵活的方式来结合这三种类型，使用我们的默认值，可以自定义，以符合一个精明的用户的利基需求。我们的方法的意义在于它产生了一个相对低维的向量，可以进行后续的资源库分析。这样的后续研究可以利用大量的ML技术：我们在这里提供了两个此类应用的概念证明。

我们部署了我们的方法，并在一个由433个资源库组成的恶意软件数据集和一个由580个资源库组成的良性数据集上研究其相似性。首先，我们通过与最先进的作品进行比较，证明了我们方法的有效性。其次，我们展示了我们的Repo2Vec是如何实现以下算法的。(a)
区分恶意软件和良性存储库，以及
(b)
确定一个有意义的分层聚类。下面简要讨论主要的结果。

a. Repo2Vec的性能优于先前的作品。在这次比较中，我们选择了迄今为止最好的方法，CrossSim，它已经被证明超过了以前的方法[8], [9], [10]。为了保持一致性，我们也遵循他们的评估方法，并使用他们的数据集与580个良性存储库。我们表明，我们的方法以93%的精度识别相似的资源库，而图1中显示的是78%。此外，我们的方法发现了近**两倍的强相似**库，并且减少了30%的假阳性，如图6所示。

b. 元数据和结构提供了重要的性能。我们评估了三类信息的信息贡献。有趣的是，如图1所示，我们可以在不使用源代码的情况下相当好地识别相似性。只使用元数据和结构会导致76%的精确度，这与之前使用源代码的最佳方法相当。

c. 应用：准确地识别恶意软件库。我们表明，我们的方法可以实现监督下的分类方法。我们专注于区分恶意软件和良性软件库，这是一个实际问题[13]。使用我们的嵌入，我们可以以98%的精度和97%的召回率识别恶意软件库，这超过了以前的方法。

d. 应用：确定一个有意义的层次结构。我们表明，我们的方法可以形成一个有意义的（无监督的）资源库的分层聚类的基础。我们表明，出现的结构与他们的目的和线路相一致。在我们的评估中，我们专注于两个层次的粒度：一个粗粒度和一个细粒度，分别有3个和26个聚类。使用基于LDA的主题提取方法，我们发现这些集群是有凝聚力的：每个集群中80%以上的资源库都有相同的关注。我们在第五节讨论聚类问题。

*我们的工作透视。*我们的方法可以被看作是在资源库分析中使用嵌入方法的第一步。事实上，它可以被看作是一个一般的框架
其中单个特征的选择可以由应用的意图驱动。例如，我们可以根据我们是否想识别：（a）抄袭或功能层面的相似性，（b）编程风格，或（c）软件的意图，来关注不同的主要特征。我们打算分享我们的方法和数据集，以促进这个方向的后续研究。

II. 背景情况

我们提供了一些关于GitHub的背景，并描述了嵌入的方法，我们在后面会扩展和使用这些方法。

A. GitHub和它的特点。GitHub是一个庞大的软件档案库，它使用户能够存储和分享代码，创造一个全球互动的社会网络。用户可以通过提出问题或分叉项目的方式在仓库中 进行合作，在这里他们复制和发展项目。用户可以关注项目，并使用 "星星 "为项目"投票"。我们在此描述GitHub仓库的 关键要素。一个仓库包含三种类型的信息（a）元数据，（b）项目目录，和（c）源代码文件，我们在下面解释。

a. 元数据。GitHub
的仓库有大量的元数据字段。最值得注意的是(a)
标题，(b) 描述。
(c)
主题，和(d)自述文件。所有这些字段都是可选的，它们是由作者提供的。Commit和issues是其他文本元数据的来源，包括关于版本库特定功能的信息。同时，有一些指标可以捕捉到版本库的受欢迎程度，包括：（a）星级，（b）分叉，和（c）手表。由于文本字段是由资源库作者提供的，它们可能是非结构化的，有噪音的，或者完全没有。

b. 源代码。它是软件库的核心元素。一个软件库包含用各种编程语言编写的软件项目，如C/C++、Java、Python等。这些源代码是存储在资源库中的软件的逻辑中心。

c. 项目目录结构。一个精心设计的软件资源库遵循一个最佳实践的目录结构，包括数据集、源代码和其他辅助文件。我们假设，这种结构在建立资源库之间的相似性方面是有用的。

B. 嵌入方法。嵌入（又称分布式表示）是一种无监督的方法，通过使用大型训练语料库上的深度神经网络将实体（如单词或图像）映射到一个相对低维的空间[14], [15]。尽管该方法是无监督的，但它依赖于理想的大型数据集，该数据集被用来 "训练"神经网络。神经网络开发了一个数据集的模型，我们可以把它看作是其实体的概率和关联性。嵌入方法已经彻底改变了几个领域的研究，如自然语言处理（NLP）[14], [15], [16], [17]。
计算机视觉[18]，图形挖掘[19]，[20]，和软件
分析[21]。

嵌入的力量是双重的：（a）它可以简化具有不同特征（包括分类）的复杂实体的表示，（b）它提供了一种量化的方法。

实体相似性是其相关向量距离的函数。一个有效的嵌入具有以下特性：(a)它给出一个固定的低维向量，(b)它确保语义上相似的对象在嵌入空间中是 "相近的"。

a. 词的嵌入：word2vec。在开创性的word2vec工作中[15]，我们将单词映射到向量，其方式是类似的单词，如 "父亲 "和 "父母"，映射到附近的向量。这种相似性是通过向深度神经网络 "输入 "大量的文档语料而建立的。换句话说，该模型通过计算一个词在一个给定的词的邻域内出现的概率来捕捉词的关联性。

b. 文档嵌入：doc2vec。doc2vec[14]是一个针对可变长度段落或文档的无监督嵌入模型。该模型将文档作为输入，并将其映射为M维嵌入向量，同时做一个代理任务，预测目标词或文档中的采样词。更详细地说，文档嵌入模型是基于单词嵌入[15]模型的。它们之间的主要区别是引入了文档id向量。与word2vec一样，有两种类型的doc2vec可用：（a）段落向量的分布式内存模型（PV-DM）和（b）段落向量的分布式词袋版本（PV-DBOW）。PV-DM类似于word2vec中的连续词袋（CBOW）模型。PV-DBOW模型类似于word2vec的skip-gram模型。文档向量是与文档的词向量同时计算的。请注意，PV-DM对于大型的、结构良好的文档表现得更好。另一方面，PV-DBOW被认为是小型和有缺陷文档的更好选择。因为它的计算速度很快。

c. 代码嵌入：code2vec.嵌入方法也被提出来用于检测代码的相似性。最近的一种方法是code2vec，它将一个任意长度的方法（或更普遍的代码片段）映射到一个M维的向量[21], [22]。code2vec方法明确使用程序结构来预测程序属性，并使用一个基于注意力的神经网络，为代码片段学习一个连续的分布式向量表示。因此，人们可以对代码片断进行比较和分组。这个过程是相当复杂的，因为它试图捕捉程序的逻辑结构和流程以及命令的顺序。例如，代码被分解成一组基于其抽象语法树的路径。神经网络同时学习：每条路径的表示方法以及如何聚合它们的一组。由于篇幅有限，我们请感兴趣的读者参考原始工作[21]。

d. 节点嵌入：node2vec。node2vec[20]是一种图嵌入方法，用于将网络中的节点映射到M维嵌入向量。该模型使用随机梯度下降法（SGD）最大限度地保留了节点的网络邻域的可能性。

更详细地说，该模型是根据节点邻域计算嵌入。首先，网络结构被转换为一组路径（节点序列），使用偏向随机的漫步式采样策略结合了每个节点的深度优先采样（DFS）和宽度优先采样（BFS）。该抽样策略有效地探索了一个给定节点的不同邻域。这些路径集可以被类比为文档中的句子。然后用word2vec[15]中提出的跳格模型对这些节点序列进行训练，得到每个节点的向量表示。关于该模型的更多细节，我们参考了原始论文[20]。

III. 建议的方法

Repo2Vec的主要思想是结合资源库的元数据、源代码和目录结构，为整个资源库提供一个嵌入表示。事实上，我们为每种类型的数据创建一个嵌入，我们称之为：(i) meta2vec代表元数据，(ii) source2vec代表源代码，(iii) struct2vec代表目录结构。我们的方法遵循这四个步骤。在前三个步骤中，我们为三类数据中的每一类创建一个嵌入向量，在第四个步骤中，我们将这些嵌入向量合并为一个资源库嵌入。Repo2Vec的流水线如图2所示。下面我们将详细解释我们方法的每一步。

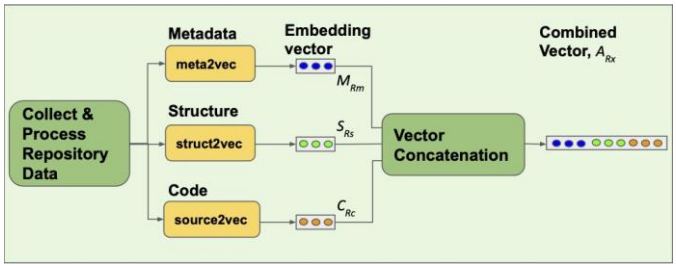

图2：Repo2Vec嵌入的概述：(a)我们为元数据、结构和源代码创建一个嵌入表示，(b)我们将它们组合成一个能捕获所有三类信息的嵌入。每一个嵌入都隐藏着重大的微妙性和挑战。

第一步。元数据嵌入：meta2vec。我们将meta2vec定义为将资源库中的所有元数据映射到一个R_M -dimensional embedding vector, M_R M。在meta2vec中，我们遵循三个步骤。首先，我们选择我们想在嵌入中 "总结 "的元数据字段。其次，我们对元数据文本进行预处理以去除噪音。最后，我们采用doc2vec方法来计算嵌入向量。meta2vec的概述见图3。

a. 字段选择。我们考虑元数据的所有字段，这些字段包含关于资源库内容的描述性信息，如标题、描述、主题（或标签）和自述文件。回顾一下，所有这些信息都是由作者提供的。有很多方法可以从每个字段中提取和组合文本信息。在这里，我们选择把每个元数据字段当作一个段落，并把它们串联起来，生成一个文件，我们对其进行如下处理。请注意，我们不考虑与资源库的受欢迎程度有关的指标，因为我们的直觉和初步结果表明，它对确定相似性的帮助不大。

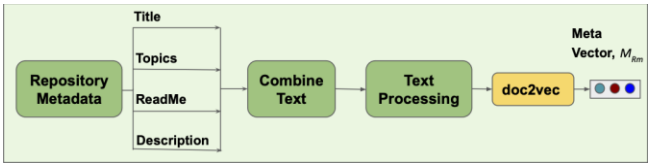

图3：meta2vec嵌入的概述：(a)我们从元数据字段中收集文本，(b)我们将它们合并成一个文件，(c)我们对文件中的文本进行预处理，(d)我们使用受doc2vec启发的方法将文件映射成一个向量。

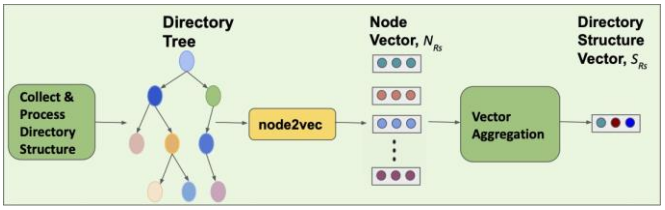

图4：我们的struct2vec嵌入概述：（a）我们提取资源库的目录树结构，（b）我们按照node2vec方法将每个节点映射成一个向量，（c）我们结合节点嵌入来创建资源库的结构嵌入。

b. 文本预处理。像任何自然语言处理（NLP）方法一样，我们首先对文本进行必要的预处理，以提高我们方法的有效性。由于资源库文本字段中的元数据通常是有噪音的，我们遵循NLP的最佳实践步骤，其中包括去除：（i）特殊字符，如'?'和'！'，（ii）不相关的单词和数字，如"URL"、"Email"、"123"，（ii）停止词。

c. 储存库元向量的生成。在这一步中，我们将资源库中的元数据映射成一个R_M -dimensional分布式矢量，M_R M。遵循doc2vec[14]方法的基本原则，我们在此根据我们的需求和限制进行调整。具体来说，由于资源库中的元数据通常是由非结构化的文本组成的，并且尺寸较小，我们采用第二节中讨论的PV-DBOW，因为它对小的文本数据集表现更好。

第二步。目录结构嵌入：struct2vec。我们将struct2vec定义为存储库目录结构与R_S -dimensional embedding vector的映射，S_R S。我们通过三个步骤来计算struct2vec。首先，我们将目录结构表示为树状表示。第二，我们用node2vec生成节点向量。第三，我们将节点向量合成为一个结构向量。图4显示了struct2vec的概况。

a. 目录树表示。GitHub中的软件仓库由一个标准的目录结构组成，包括必要的数据文件和源代码文件。我们考虑目录结构，并将其转化为树状表示法，以便在其上实现node2vec。注意，为了消除目录或文件名在映射中的影响，该表示法不包括树中的目录或文件名。

b. 节点向量的生成。在这一步中，我们将树中的所有节点映射成一个R_S -dimensional节点嵌入向量，N_R S。按照node2vec的特性，首先，我们使用偏向随机行走的抽样策略将树转换成一组路径，以包括一个节点的不同的邻接节点。然后，我们在这些路径上应用跳格模型，得到所有节点的向量。

c. 储存库目录结构向量的生成。我们通过合成树中的节点向量，N_R S，计算存储库目录结构嵌入向量，S_R S。我们采用逐列聚合的方法，将这些节点向量合成为一个单一的向量。为了做到这一点，我们采用了六个聚合函数：平均值、模式、最大值、最小值、总和、标准差来计算结果向量中某一列的值。

第3步。源代码嵌入：source2vec。我们将source2vec定义为一种嵌入方法，将资源库中的源代码表示为一个R_C -dimensional的嵌入向量。在source2vec中，我们采用了Java方法嵌入技术和第二节中讨论的15.3M方法的训练模型。我们在source2vec中遵循三个步骤。首先，我们为资源库中的源文件中的每个方法计算R_C -dimensional方法代码向量。其次，我们将这些方法向量汇总到一个单一的R_C -dimensional文件代码向量中。最后，我们通过另一级的向量聚合，为所有的源文件计算出最终的R_C -dimensional资源库代码向量。我们的方法的流水线如图5所示，下面将详细讨论。

a. 方法代码矢量的生成。一个软件资源库可能有多个源代码文件和其他文件。首先，每个源文件被分解为其方法。接下来，方法被预处理成AST路径和上下文向量，作为code2vec模型的输入。该模型将每个方法映射成一个R_C -dimensional embedding code vector, MC_R C。然后，这些方法向量被传递到下一阶段的管道中，被聚合成一个单一的向量。

b. 文件代码向量的生成。在生成文件中的方法代码向量MC_R C后，现在的任务是将它们汇总成一个R_C -dimensional file code vector, FC_R C。我们遵循一些列的聚合函数。我们研究的聚合函数是平均值、模式、最大、最小、总和和标准差。在这个过程中，流水线创建了一个单一的文件代码向量，FC_R C，并将其传递给下一个阶段，以创建一个单一的资源库向量。

c. 储存库代码向量的生成。在这个阶段，source2vec将版本库中所有源代码文件的R_C -dimensional file code vector, FC_R C聚合为一个单一的R_C -dimensional repository code vector, C_R C。该管道遵循与上一步骤相同的程序，通过逐列聚合函数得到版本库代码向量。

第4步。Repo2Vec。仓库嵌入。我们提出Repo2Vec，用三种信息源的特征在嵌入向量中展示GitHub仓库：元数据、源代码和项目目录结构，遵循图2所示的管道。在这个步骤中，我们将

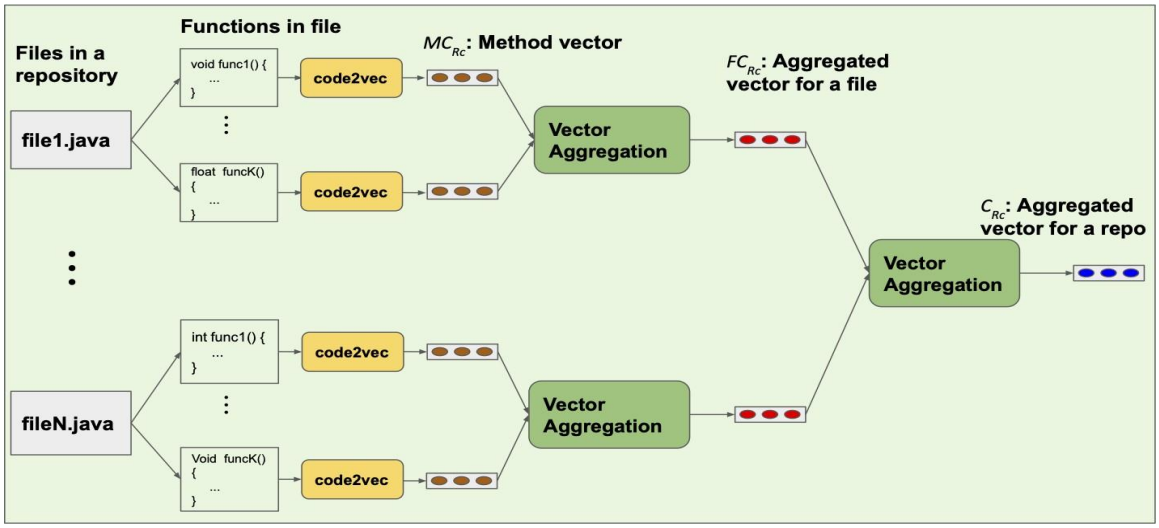

图5：我们的source2vec嵌入的概况：（a）我们从每个源文件中提取函数（方法），（b）我们嵌入每个函数。
(c) 我们将每个函数嵌入结合起来，为每个文件创建一个嵌入，以及(c)我们将每个文件嵌入集合起来，为资源库创建源代码嵌入。

元数据向量M_R − M，目录结构向量S_R S，以及源代码向量C_R C进入存储库向量A_R x。

将每种信息类型的向量结合起来是一个重要的挑战。许多方法都有两种类型的应用：(a)将数值合并成一个单一的向量，使用总和、平均数或中位数等，以及(b)将向量连接起来以创建一个 "较长"向量。在这两种方法中，可以考虑加权和归一化，以确保
"公平"。在这里，我们选择使用串联的方法，如下所示。

$$A_R \, \mathrm{x} = w_M * M_R \, \mathrm{M} + w_S * S_R \, \mathrm{S} + w_C * C \, C_R \quad (1)$$

其中，w_M 、 w_S 和w_C 分别是元向量M_R M、结构向量S_R S和源代码向量C_R C的权重，这些权重的范围是[0, 1]。

IV. 实验和评估

我们使用真实数据评估Repo2Vec的有效性，并回答两个问题。

Q.1:每种信息类型的影响是什么？我们想量化三种信息类型在确定相似性方面的效果和贡献。

问题2：Repo2Vec与现有技术相比如何？我们将我们的方法与CrossSim[12], [11]进行比较，后者可以说是最先进的方法，并被证明优于以前的方法[8], [10], [9]。

A. 实验设置

我们介绍了数据集和我们的评估方法。

1. 数据集。我们在评估中考虑了两个数据集：（a）良性软件库的数据集D − ben，这在之前的工作中使用过[11]，[12]；（b）恶意软件库的数据集D mal，由之前的软件库分析研究收集[13]。

a. 良性资源库D

ben：该数据集由GitHub的580个Java资源库组成，在早期的研究中使用过。

介绍CrossSim[11]。我们选择这个数据集是为了与CrossSim进行公平和可重复的比较。该数据集横跨各种软件类别，例如。PDF处理器、JSON解析器、对象关系映射项目、Spring
MVC相关工具、SPARQL和RDF、Selenium测试、Elastic搜索、Spring MVC、Hadoop和音乐播放器。

b. 恶意软件存储库D mal。这个数据集包括
433个Java恶意软件存储库。该数据集由SourceFinder项目[13]提供，其目标是识别和提供恶意软件源代码库。在这里，我们只选择了Java语言库，这也是CrossSim方法的重点。该软件库在各恶意软件家族中的覆盖面相当广泛，包括。僵尸网络、键盘记录器、病毒、勒索软件、DDoS、间谍软件、漏洞、垃圾邮件、恶意代码注入、后门程序和木马。

2. 基于查询的评价。为了一致性和公平性，我们遵循先前的工作[11]的评价方法和相似性指标。我们通过使用相似性查询来进行评价，具体如下：一个给定的资源库，我们要确定其五个最相似的资源库。

*a. 查询集Q ben。*为了与CrossSim兼容，*Q ben由与*CrossSim相同的50个资料库查询集组成。该查询集横跨多个领域
例如，SPARQL和RDF、Selenium测试、Elastic搜索、Spring MVC、Hadoop和音乐播放器。

*b. 查询集Q mal。*对于D级恶意软件数据集，我们通过均匀地随机选择50个存储库来创建一个查询集。该查询集包括各种恶意软件系列，如键盘记录器、僵尸网络、DDoS、勒索软件、病毒、后门、木马等。

3. 地面真相的生成。我们通过人工评估为每个数据集建立基础真相，并遵循先前工作中使用的评分框架[11]。也就是说，我们使用四类分数来标示相似度的水平。

- 第4类。强相似（**SS**）库。
- 第三类：弱相似（**WS**）库。
- 第2类：弱异性（**WD**）存储库。
- 第1类：强相异性（**SD**）库。

为了保持一致性，我们遵循以前的研究[11]的惯例：类别3或4的存储库被认为是（足够）相似或真阳性。反之，1或2类的资源库被认为是不相似的或假阳性。在评估中，我们选择使用专家，与Mechanical Turk平台相比，专家在处理高技术问题时更可靠 [23]。具体来说，我们招募了三名具有至少3年Java编程经验的计算机科学研究员。评估者得到了目标资源库和每个查询的5个资源库的响应。注意，每个响应中的5个资源库是随机的，以避免引入偏见。评价者在四类分数中给响应中的每个资源库分配一个分数。为了校准他们的标准，评价者被提供了背景和信息。第一位和第二位评价者独立地给答复中的每个存储库打分。后来，第三位评价者作为法官，重新检查并最终确定他们的分数，如果他们的分数对一个查询来说是不一样的。

4. 评价指标。为了保持一致性，我们采用了相关工作中使用的指标[11]，我们在下面进行描述。

a. 成功率。如果返回的一个或多个资源库与上述的相似性定义相似，我们就说查询的答案是成功的。成功率是指成功查询的百分比。

b. 精度。精确率是指返回的资源库中与查询资源库相似的百分比。我们按照以下公式计算精度。

$$\text{精度} = \frac{SS + WS}{SS + WS + WD + SD} \quad (2)$$

c.真阳性和假阳性。按照标准的定义，一个查询集的真阳性是返回的相似库的总数，而假阳性是答案中非相似库的数量。

d.排名顺序相关（ROC）。我们再次使用先前工作中引入的指标对查询的排名答案的质量进行量化。其直觉是要"奖励
"一个返回高度相似的存储库排名较高的算法。为了量化这一点，我们计算了广泛使用的Spearman等级相关系数r[24]，其定义为：。

$$r = 1 - \frac{6\sum(d_i)^2}{n(n^2 - 1)} \quad (3)$$

其中r是系数，d_i是每个存储库的两个排名之间的差异，n是排名的存储库数量。系数的范围是$[-1, 1]$，1表示完全一致，-1表示不一致。
两个排名。

评论。考虑到我们制定查询的方式，使用召回率在这里就不那么重要了：我们要求算法只报告前五个最相似的存储库。拟定一个查询我们希望这些方法能够返回所有类似的资源库，这具有挑战性，原因有二。首先，我们需要一个既定的基础真理，因为人工验证将是劳动密集型的。第二，没有绝对的方法来定义什么是"足够相似
"的资源库，而相对相似性更容易定义。

B. 部署Repo2Vec

我们使用Python3.6软件包来实现我们的方法，我们在第三节中描述了这一方法。TensorFlow2.0.0, gensim PV-DBOW doc2vec。我们讨论了一些实施细节和参数选择。

选择嵌入的维度。我们选择128作为R_M，R_S，和R_C的嵌入向量维度，因为成熟的嵌入技术[15], [14], [21]。[20]推荐这个数字，以便在计算成本和有效性之间取得平衡。为了公平起见，我们对每种类型的信息向量使用相同的维数。将这三个向量串联起来就形成了一个具有R_x=384维度的Repo2Vec向量。上述选择给出了良好的结果，我们将在后面看到。在未来，我们将探索不同向量维度的效果。

通过权重选择探索解决方案的空间。方程1中的权重使我们有能力控制每种信息类型的
"贡献"。在这里，我们重点关注以下权重组合，这些组合产生了三种衍生算法。(a) **Repo2Vec M**仅使用元数据，权重为$w_M = 1, w_S = 0, w_C = 0$；(b) **Repo2Vec MS**使用元数据和结构，权重为$w_M = 1, w_S = 1, w_C = 0$；以及，(c) **Repo2Vec All**使用所有三种信息，权重为$w_M = 1, w_S = 1, w_C = 1$。

换句话说，我们探讨了权重的影响，但是是以一种粗略的方式。在未来，我们打算探索非整数的
权重组合。总的来说，我们的结果表明，相等的权重似乎效果很好，但一个精明的用户可以对它们进行定制，以实现利基问题的最佳性能。

计算相似性。有许多不同的方法来计算嵌入空间中的相似性，即它们在该空间中的距离的倒数。在这里，我们使用广泛使用的余弦相似度，它通常被推荐用于高维空间[25]，并且在这里也能产生很好的结果。**选择正确的聚合函数，将多个向量聚合成一个向量。**正如我们在第三节中所看到的，我们引入了六个逐列聚合函数来将向量聚合成一个向量。我们发现，平均聚合函数的表现比其他函数好。在更详细的尾部，我们评估了所有聚合函数的性能。
平均值、最大值、最小值、模式、总和和标准差。我们发现，使用平均聚合的嵌入功能对Dben数据集显示出最高的93%精度，对Dmal数据集显示出95%的精度。最大聚合函数对良性数据集和恶意软件数据集显示了第二好的结果，分别为88%和91%的精度。其他聚合函数对这两个数据集显示的精度相对较低。在剩下的工作中，我们使用平均聚合函数。

方法	D本数据集		D 恶意数据集	
	成功率	精度	成功率	精度
Repo2Vec M	100%	62%	100%	67%
Repo2Vec MS	100%	76%	100%	82%
Repo2Vec All	**100%**	**93%**	**100%**	**95%**

表一:我们对Repo2Vec的三种变体的性能比较。使用所有三种信息类型（元数据、结构和源代码）提供了明显更好的结果。

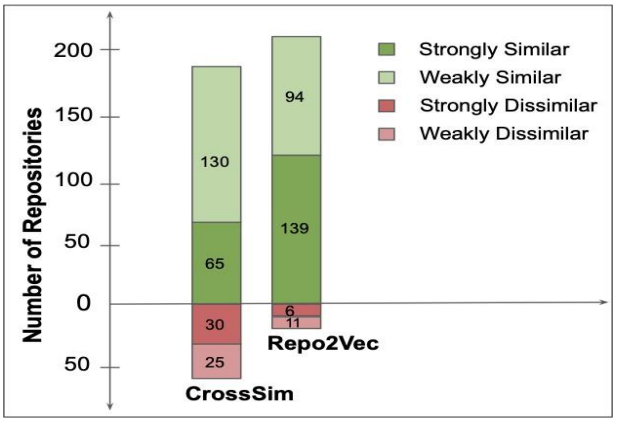

图6：Repo2Vec的表现明显优于CrossSim：它发现的强相似库几乎是CrossSim的两倍，误报率降低了30%。

C. 评价

我们以两种方式评估Repo2Vec。首先，我们评估了每种类型的信息对性能的影响。其次，我们将我们的方法与CrossSim[11]进行比较，后者是目前最先进的方法。

a. 信息类型的影响。我们通过比较三种变体的性能来评估信息类型的影响。Repo2Vec M、Repo2Vec MS和Repo2Vec All，这是我们之前定义的。我们在表I中报告了我们的三种Repo2Vec变体和两个数据集的结果。这一评估导致了两个主要的观察。

观察1：使用所有三种数据类型提供了明显更好的性能。在表中，我们看到Repo2Vec All实现了93%和95%的精度，而只使用metada和结构信息时，精度为76%和82%。

观察2：元数据和结构提供了相当好的结果。虽然Repo2Vec All表现最好，但Repo2Vec MS的表现也相当不错，特别是如果我们在表二所示的相同的良性数据集和查询集上与CrossSim进行比较。请注意，与分析代码相比，使用元数据和结构的计算工作量要小得多。

b. 将Repo2Vec与最新技术水平进行比较。我们将最佳配置Repo2Vec All与Cross-Sim在良性数据集Dben的成功率、精确度、置信度和ROC方面进行比较。我们发现，Repo2Vec

方法	成功率	精度	斯佩尔曼的系数(r)
交叉模拟	100%	78%	0.23
Repo2Vec 全部	100%	93%	0.59

表二：Repo2Vec和CrossSim在Dben数据集的相似性方法比较中表现更好。

在精确度和ROC方面优于CrossSim，并具有与CrossSim相同的成功率。

观察3：Repo2Vec：精度更高，排名更好。结果列于表二。虽然**CrossSim**在成功率方面表现不错，但其78%的精度与Repo2Vec All的93%的精度相比明显偏低。另外，Repo2Vec All识别的类似资源库的排名也比**CrossSim**好。我们发现，Repo2Vec All的ROC=0.59，而**CrossSim**的ROC=0.23，这进一步说明Repo2Vec All在计算资源库之间的相似度方面更胜一筹。

观察4：Repo2Vec提供了更好的质量结果。鉴于我们有四类相似性，我们评估结果的质量如下。我们在图6中画出了每个类别中每种方法所返回的资源。仅考虑第4类（强相似性），Repo2Vec All识别出的此类资源库多出近100%！而CrossSim识别出的此类资源库则多出5倍。同样地，CrossSim在强相似性类别中报告的资源库多出5倍。

总之，我们的比较表明，Repo2Vec的性能优于CrossSim。表二和图六对评估进行了总结。此外，CrossSim的表现也优于其他相关作品RepoPal、CLAN和MUDABLUE[11]。

V. 案例研究

我们发现，Repo2Vec

在这一节中，我们想展示Repo2Vec如何在考虑无监督和有监督技术的情况下促进特定应用的资源库挖掘研究。我们考虑两个可能的案例研究：a）将资源库分类为良性或恶意的，b）对一组资源库进行聚类。

A. 识别恶意软件存储库

我们展示了我们的Repo2Vec在一个超视距分类问题中的作用，这是实践者感兴趣的问题[13]，[26]，[27]。问题是要识别一个软件库是否包含恶意软件或良性代码。我们评估了我们的方法的有效性，我们还将其与最先进的方法[13]进行了比较。

我们创建了一个由D ben的580个良性软件库和D mal的433个恶意软件库组成的数据集，在第四节中进行了讨论。

方法	准确度	精度	召回	F1得分
源头搜索	90%	89%	99%	94%
呼叫中心	97%	98%	96%	97%

表三：Repo2Vec在恶意软件库分类中优于SourceFinder

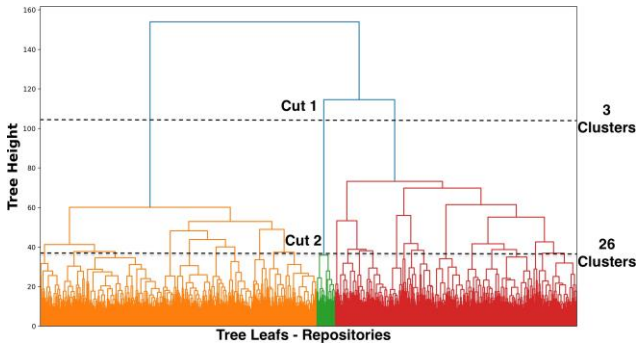

图7：恶意软件库的分层聚类。横线1切入3个不同的存储库群，横线2切入26个不同的存储库群。

使用我们的Repo2Vec，我们确定每个资源库的嵌入向量。对于分类，人们可以使用大量的ML方法。在这里，我们使用Naive
Bayes，它被广泛用于NLP分类问题[28]，更重要的是，它也被最近的SourceFinder研究[13]使用。通过这种选择，我们想把重点放在
在与SourceFinder分类法的比较中，更多的是对特征的影响。我们实现了SourceFinder分类器，并将其应用于我们的数据集。

我们使用10倍交叉验证来评估分类性能。结果显示在表三。我们的模型对恶意软件和良性软件库的分类有98%的预审率和96%的召回率，这显然超过了SourceFinder[13]以前的恶意软件库分类研究。

B. 分层聚类

在这里，我们展示了我们的方法是否能够导致一个有意义的存储库聚类，为无监督的解决方案创造基础。我们考虑了我们的两个数据集，D mal和D ben数据集的联合，总共有1013个存储库。

首先，我们在所有资源库上应用Repo2Vec，得到嵌入向量。其次，我们在存储库的向量上应用广泛使用的聚类分层聚类（AGNES）[29]。显然，有许多不同的聚类技术，但请注意，我们的目标是展示能力，而不是提出一种聚类方法。我们在图7中展示了所得的分层聚类。

这种聚类的意义有多大？评估分层聚类的效果是很有挑战性的，它可能取决于一个研究的具体重点。一个相关的问题是，我们应该关注哪些层次的颗粒度。我们提供间接证明，我们的聚类提供了有意义的结果。

考虑到两个层次的粒度。我们在两个不同的粒度水平上分析了我们的分层聚类，在图7中用两条横线表示。第一条线（Cut
1）对应的是**粗粒度水平**，产生三个大的聚类。第二条线（Cut 2）对应的是**细粒度水平**，产生26个较小的聚类。

群体没有。	数量回报率	主导者回购家族	群体没有。	数量回报率	主导者回购家族
1	25	DDoS	14	10	病毒
2	27	安卓键盘记录器	15	58	木马和间谍软件
3	42	后门	16	33	REST API
4	32	蠕虫	17	48	Hadoop
5	44	安卓僵尸网络	18	36	JSON解析器
6	55	安卓恶意软件	19	45	音乐队员
7	31	Rootkit	20	71	SPARQL
8	24	爪哇键盘记录器	21	146	弹性搜索
9	32	勒索软件	22	54	宗旨关系图谱
10	24	白帽子黑客攻击	23	27	PDF处理器
11	15	恶意的代码注入	24	25	图-辅助搜索c
12	8	安卓木马	25	31	硒
13	6	安卓后门	26	56	春天MVC

表四：精细级别的聚类。 的概况。 26 贮藏室

使用主题提取方法的聚类。聚类的颜色与图7的颜色相似。

群体没有。	数量回报率	群体类型	集群描述
1	433	恶意软件	D型恶意软件库
2	33	良性	第16组来自细粒度与REST API存储库
3	547	良性	D本资料库。

表五:粗略级聚类：三个聚类的概况。聚类的颜色与图7的颜色相似
.

我们在图7中详细说明了我们如何选择两个切口。首先，我们选择Cut
1，看看聚类是否能将恶意软件与良性软件库区分开来。其次，我们以优化集群数量的方式选择切割2。一个常用的方法是弯头法[30]。曲线的肘部或膝盖是聚类数量与平方误差之和（SSE）图中的一个分界点，在那里增加聚类数量显示出收益递减。图8显示，肘部位于K=26个群集左右，这就是我们选择Cut 2的原因。

我们的目标是在这两个层次的颗粒度上对识别的集群进行剖析。结果显示在表四和表五中。

1. 精细级别的集群剖析。我们想在这个层面上评估26个群组的性质和凝聚力。我们从每个集群的重点方面提取其概况，我们在表四中列出了结果。我们的剖析包括两个步骤。(a)我们确定集群的主导关键词，(b)我们评估其资源库与概况集群的吻合程度。更详细地说，我们使用Late nt Dirichlet
Analysis（LDA）主题建模[31]对集群主题进行识别。

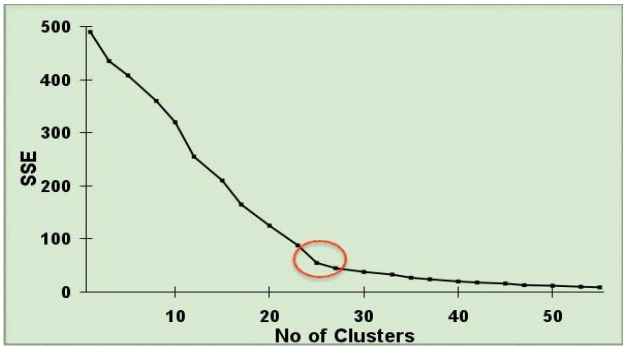

图8：确定最佳的聚类数量。红圈处显示了平方误差之和（SSE）的递减回报。

每个资源库的元数据。请注意，我们使用的是随机选择的集群中一半的存储库的子集。第二，我们要在所有的候选主题中找出最主要的主题。最主要的主题是出现在集群中最多资源库的主题。我们在表四中报告该主题。该集群的凝聚力很强：至少有80%的被检查的资源库明显属于该集群的家族成员。最后，作为一个额外的可选步骤，我们手动调查资源库，以验证配置文件的准确性。

这个过程给我们提供了具有凝聚力和"重点"的集群。大多数集群包含来自狭义的恶意软件或良性软件家族的软件库，如Android Botnet、Keyloggers、Trojan、DDoS、Backdoor、Hadoop、Json解析器、Elastic Search和Spring MVC。

我们提供了一个可以从这里提取的洞察力的迹象。有趣的是，最大的恶意软件集群（集群15）有58个存储库，包含来自木马和间谍软件家族的存储库。木马恶意软件程序与间谍软件类似，只是它被包装成另一个程序。这一观察可以产生以下假设：木马和间谍软件的共同点是否比我们想象的要多？

2. 粗略的集群分析。最重要的观察结果是，这个级别的三个集群正确地对应于不同的软件领域，如表五所示。我们发现以下集群：（a）D级恶意软件库。(b)D本，良性软件库，和(c)REST API相关的良性软件库，对应于细粒度聚类中的第16组。无监督聚类将恶意软件和良性软件库分开的事实表明，恶意软件和良性软件是不同的。唯一的例外似乎是REST API集群16，如果我们创建了两个集群的分解，它将与恶意软件库捆绑在一起。我们认为，REST API软件库似乎类似于ddos和僵尸网络恶意软件（打开和监听端口等）。

VI. 讨论

在这一节中，我们讨论了我们研究的范围、扩展和局限性。

a. Repo2Vec的局限性是什么？由于Repo2Vec是一个全面的方法，有来自三个不同来源的数据，即使每个数据源都不存在，它也能执行。然而，我们认为，具有回避元数据和混淆源代码的非结构化软件库可能会愚弄Repo2Vec。在这种情况下，以前的工作可能会有更好的表现，因为这些工作主要依赖于存储库的图连接。

b. 我们的方法是否可以推广到其他编程语言库？我们的方法对于所有的编程语言都是可以通用和扩展的，尽管准确度可能有所不同。首先，code2vec[21]可以扩展到其他编程语言，而且研究人员似乎有计划扩展到其他语言。其次，两种信息类型，元数据和结构，是相当独立于编程语言的。此外，从表一中，我们可以看到，即使只使用这两种信息类型，我们也可以取得相当好的性能。

c. 如果元数据的质量很低或有误导性，会发生什么？如果元数据变得不可靠，我们可以降低它在我们算法中的权重。同时，我们发现，开发者有提供高质量元数据的内在动机。首先，这些资源库是开发者职业角色的一部分，是一个人的职业组合或简历的一部分。其次，这些资源库是公开的，因此有意向使它们既容易找到又容易使用。拥有一个受欢迎的资源库的炫耀权利是提供信息元数据的强大动力。因此，这类资源库的数量往往很低。我们在Dben的580个资源库中只有1个（0.17%），在Dmal的433个资源库中只有3个（0.69%）元数据为空。另外，由于Repo2Vec是一个综合的方法，有三个信息源的数据，即使元数据不可用，它也会有足够的表现。

d. 为什么GitHub搜索不足以识别类似的仓库？GitHub只允许根据关键词来检索仓库。虽然非常有用，但GitHub的查询能力并没有回答我们在这里要解决的问题。首先，它不支持实例查询。"找到与此仓库最相似的仓库"。第二，它没有提供测量一对仓库之间的相似性的能力，也没有根据与一个给定仓库的相似性对一组仓库进行排名。第三，该服务似乎没有使用源代码，正如我们所看到的，它提供了重大改进。

e. 我们的数据集有代表性吗？这是任何测量研究中典型的难以回答的问题。我们试图通过两个声明来回答这个问题。首先，我们用先前著名的研究[11]、[12]所使用的580个资源库的相同数据集（D本）来评估我们的方法。这个数据集试图包括表四中所列的十个不同家族的资源库。其次，我们的Dmal数据集包括同一表格中列出的13种类型的恶意软件家族。在未来，我们打算在我们的数据集中收集更多的存储库，并包括更多的编程语言。关键的瓶颈是创建groundtruth。

f. 我们应该考虑流行度量吗？到目前为止，我们还没有考虑到存储库的流行度量。

诸如星星、手表和分叉的数量等。虽然我们打算研究我们能从这些指标中提取什么信息，但我们认为它们主要有助于找到有代表性或有影响力的存储库。我们的初步分析表明，人气并不能提供信息
储存库的类型。作为一个概念的证明，我们可以考虑一个初始仓库和一个分叉仓库：它们很可能几乎是相同的，但它们的流行度量可以有很大的不同。

VII. 相关作品

在过去的几年里，研究软件库之间的相似性已经获得了极大的关注。大多数研究与我们的方法不同，因为(a)
他们没有纳入软件库中存在的所有类型的数据，(b)
他们没有提出一个保持软件库的元数据、源代码和结构的语义的特征向量，(c)
他们的方法不适合其他ML分类任务，如软件库家族分类、恶意软件和良性软件库分类等。下面我们简要讨论一下相关的工作。

a. 软件相似性计算。之前的软件相似性计算研究主要可以根据他们使用的数据分为三类：（a）高层次的元数据[7]，[5]，[6]，[8]，（b）低层次的源代码[9]，[10]，和(c)
高层和低层数据的结合[11]，[12]。在早期的研究中[7]，作者利用资源库标签来计算用不同语言编写的资源库之间的相似度。掌握了资源库中标签的权重，他们创建了特征向量并应用余弦相似度来计算相似度。后来，[5]提出了一种图书馆推荐方法，LibRec，使用关联规则挖掘和协同过滤技术。它搜索相似的库来为开发者推荐相关的库。另一项工作[6]提出了SimApp来识别具有类似语义要求的移动应用程序。最近的一种方法，RepoPal[8]，利用GitHub存储库的readme文件和star属性来计算相似度。在两个存储库之间。

另一方面，MUDABLUE[9]是第一个使用源代码的潜在语义分析（LSA）对软件库进行分类的自动方法。考虑到源代码是纯文本，他们创建了一个标识符-
软件矩阵，并对其应用LSA来计算相似度。后来，另一项研究[32]对软件库进行了分类，在源代码上应用了Lent Dirichlet
Allocation（LDA）。最近一项名为CLAN[10]的研究通过将源代码文件表示为术语-
文档矩阵（TDM）来计算资源库之间的相似性，其中每个类别代表一行，资源库为列。

最后，最近的一项研究[11]，[12]提出了CrossSim，这是一种基于图的相似性计算方法，使用高层次的星形属性和资源库中源代码文件的API调用参考。利用相互关系，他们将一组资源库表示为一个图，并从图中计算出一个给定资源库的类似资源库。然而，他们的工作受到了外部库调用的限制，因为相似性在很大程度上取决于它。另一项研究[33]证实，CrossSim可以在内部实现类似功能的同时，根据外部API的使用情况来识别异同。

b. 嵌入方法。自然语言处理（NLP）领域的最新进展为特征表示开辟了一种全新的方式，即基于神经网络的离散对象特征学习方法。首先，word2vec[15]的引入，为来自非常大的语料库的连续向量表示，铺平了道路。后来，另一项名为doc2vec[14]的研究引入了可变长度段落或文档的分布式表示。最近，嵌入的概念被其他领域所共享，并在有效的特征表示中获得了巨大的成功，如图嵌入[20]，[19]，主题嵌入[34]。
鸣叫嵌入[35]，以及代码嵌入[21]，[36]，[37]。
[38].

VIII. 结论

我们提出Repo2Vec，这是一种利用三类信息源的数据，在嵌入向量中表示一个资源库的方法。(a) 元数据，(b) 存储库结构，和
(c)
存储库中可用的源代码。主要的想法是将这三类信息的嵌入表示聚合起来。

我们的工作可以归纳为以下几点。

1) **一个高效的嵌入。**Repo2Vec是一种全面的嵌入方法，它使我们能够以93%的精度确定类似的存储库。
2) **提高了技术水平。**我们的方法比最好的方法CrossSim在精度上要高出15%。此外，它还发现了近两倍的强相似库，并减少了30%的假阳性。
3) **促进了对恶意软件的识别。**我们的方法能够以98%的精度对恶意软件和良性软件库进行分类，超过了以前的研究。
4) **使得有意义的聚类成为可能。**我们的方法确定了资源库的树状层次结构，与它们的目的和脉络非常吻合。

在未来，我们首先计划用更大的数据集和更广泛的地面真实数据集来扩展这项工作。事实上，我们想帮助开发一个社区范围的基准，以促进进一步的研究。其次，我们希望将我们的工作扩展到其他编程语言，这主要取决于为其他语言开发code2vec能力。看看不同的语言是否可以像我们在这里对Java所做的那样嵌入表示，这将是很有趣的。

最后，我们打算将我们的代码和数据集开源，以最大限度地扩大我们工作的影响，并促进后续研究。

参考文献

[1] A.Mockus, D. Spinellis, Z. Kotti, and G. J. Dusing, "通过社区检测方法识别的一整套相关git存储库

based on shared commits," in *IEEE International Working Conference on Mining Software Repositories* , 2020.

[2] D.Spinellis, Z. Kotti, and A. Mockus, "A dataset for github repository deduplication:扩展描述，"在*IEEE国际工作会议*上*关于挖掘软件库* ，2020年。

[3] M.Gharehyazie, B. Ray, and V. Filkov, "有些来自这里，有些来自那里。github中的跨项目代码重用，"在*2017年IEEE/ACM第14届挖掘软件库国际会议（MSR）*。IEEE，2017，pp.291-301。

[4] A.Mockus, "开源软件中的大规模代码重用"，*在第一届FLOSS研究与开发新趋势国际研讨会（FLOSS'07: ICSE Workshops 2007）*。IEEE，2007，pp. 7-7.

[5] F.Thung, D. Lo, and J. Lawall, "Automated library recommendation," in *2013 20th Working conference on reverse engineering (WCRE)*.IEEE，2013年，第182-191页。

[6] N.Chen, S. C. Hoi, S. Li, and X. Xiao, "Simapp:一个通过在线内核学习检测类似移动应用的框架，"在*第八届ACM网络搜索和数据挖掘国际会议论文集*，2015年，第305-314页。

[7] F.Thung, D. Lo, and L. Jiang, "Detecting similar applications with collaborative tagging," in *2012 28th IEEE International Conference on Software Maintenance（ICSM）*.IEEE，2012年，第600-603页。

[8] Y.Zhang, D. Lo, P. S. Kochhar, X. Xia, Q. Li, and J. Sun, "Detecting similar repositories on github," in *2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering（SANER）*.IEEE，2017，pp.13-23。

[9] S.Kawaguchi, P. K. Garg, M. Matsushita, and K. Inoue, "Mudablue:Mudablue: An automatic categorization system for open source repositories," *Journal of Systems and Software*, vol.79, no.7, pp.939-953, 2006.

[10] C.McMillan, M. Grechanik, and D. Poshyvanyk, "Detecting similar software applications," in *2012 34th International Conference on Software Engineering (ICSE)*.IEEE, 2012, pp.364-374.

[11] P.T. Nguyen, J. Di Rocco, R. Rubei, and D. Di Ruscio, "Crosssim: exploiting mutual relationships to detect similar oss projects," in *2018 44th Euromicro conference on software engineering and advanced applications（SEAA）*.IEEE，2018，第388-395页。

[12] --，"评估github repositories相似性的自动方法"，*《软件质量杂志》*，第1-37页，2020。

[13] M.O. F. Rokon, R. Islam, A. Darki, E. E. Papalexakis, and M. Faloutsos, "Sourcefinder:从github的公开可用库中寻找恶意软件的源代码，"在*第23届攻击、入侵和防御研究国际研讨会（{RAID}2020）上*，2020年，第149-163页。

[14] Q.Le and T. Mikolov, "Distributed representations of sentences and documents," in *International conference on machine learning*, 2014, pp. 1188-1196.

[15] T.Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," *Advances in neural information processing systems*, Vol. 26, pp.3111-3119, 2013.

[16] J.Pennington, R. Socher, and C. D. Manning, "Glove:全局向量的单词表示，"在*2014年自然语言处理经验方法会议（EMNLP）论文集*，2014年，第 1532-1543。

[17] M.E. Peters, M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee, and L. Zettlemoyer, "Deep contextualized word representations," *arXiv preprint arXiv:1802.05365* , 2018.

[18] S.Kottur, R. Vedantam, J. M. Moura, and D. Parikh, "Visual word2vec（visw2v）。使用抽象场景学习有视觉基础的单词嵌入，"在2016年*IEEE计算机视觉和模式识别会议论文集中*，第4985-4994页。

[19] A.Narayanan, M. Chandramohan, R. Venkatesan, L. Chen, Y. Liu, and S.Jaiswal, "graph2vec:学习图形的分布式表示，" *arXiv预印本 arXiv:1707.05005* , 2017。

[20] A.Grover和J. Leskovec，"node2vec:可扩展的网络特征学习，"*《第22届ACM SIGKDD知识发现与数据挖掘国际会议论文集》*，2016年，第855-864页。

[21] U.Alon, M. Zilberstein, O. Levy, and E. Yahav, "code2vec:Learning distributed representations of code," *Proceedings of the ACM on Programming Languages*, vol. 3, no.POPL，第1-29页，2019年。

[22] R.Compton, E. Frank, P. Patros, and A. Koay, "Embedding java classes with code2vec:来自变量混淆的改进"，*论文集*

of the 17th International Conference on Mining Software Repositories, 2020.

[23] J.Gharibshah, E. E. Papalexakis, and M. Faloutsos, "休息。用于识别和分类安全论坛中的用户指定内容的线程嵌入方法，"*《国际AAAI 网络和社会媒体会议论文集》*，第14卷，2020，第217-228页。

[24] C.斯皮尔曼，"两个 之间关联的证明和测量"。*Appleton-Century-Crofts*, 1961.

[25] G. Sidorov, A. Gelbukh, H. Go´mez-Adorno, and D. Pinto, "Soft similarity and soft cosine measure: Similarity of features in vector space model, " *Computacio´n y Sistemas*, vol. 18, no.3, pp. 491-504, 2014.

[26] M.Soll和M. Vosgerau, "Classifyhub: an algorithm to classify github repositories," in *Joint German/Austrian Conference on Artificial Intelligence (Ku¨nstliche Intelligenz)*.Springer，2017，第373-379页。

[27] Y.Zhang, F. F. Xu, S. Li, Y. Meng, X. Wang, Q. Li, and J. Han, "Higit-class:关键词驱动的github资源库分层分类，"在*2019年IEEE国际数据挖掘会议（ICDM）*。IEEE， 2019，第876-885页。

[28] S.Xu, "Bayesian na¨ve bayes classifiers to text classification," *Journal of Information Science*, vol. 44, no. 1, pp. 48-59, 2018.

[29] F.Murtagh和P. Legendre, "Ward的分 层 聚类法：哪些算法实现了Ward的标准？"*杂志 of classification*, vol. 31, no.3, pp. 274-295, 2014.

[30] T.M. Kodinariya and P. R. Makwana, "Review on determining number of cluster in k-means clustering," *International Journal*, vol. 1, no. 6, pp.90-95, 2013.

[31] D.M. Blei, A. Y. Ng, and M. I. Jordan, "Latent dirichlet allocation," *Journal of machine Learning research*, vol. 3, no.Jan, pp. 993-1022, 2003.

[32] K.Tian, M. Revelle, and D. Poshyvanyk, "Using latent dirichlet allocation for automatic categorization of software, " in *2009 6th IEEE International Working Conference on Mining Software Repositories*.IEEE，2009，第163-166页。

[33] A.Capiluppi, D. Di Ruscio, J. Di Rocco, P. T. Nguyen, and N. Ajienka, "Detecting java software similarities by using different clustering tech- niques, " *Information and Software Technology*, vol. 122, p. 106279, 2020.

[34] L.Niu, X. Dai, J. Zhang, and J. Chen, "Topic2vec: learning distributed representations of topics," in *2015 International conference on asian language processing（IALP）*.IEEE, 2015, pp.193-196.

[35] B.Dhingra, Z. Zhou, D. Fitzpatrick, M. Muehl, and W. W. Cohen, "Tweet2vec:基于字符的社交媒体分布式表征 dia，" *arXiv预印本arXiv:1605.03481*，2016。

[36] T.Hoang, H. J. Kang, D. Lo, and J. Lawall, "Cc2vec:代码变化的分布式表示，"*在ACM/IEEE第42届 国际软件工程会议论文集*，2020年，第518-529页。

[37] H.J. Kang, T. F. Bissyande´, and D. Lo, "Assessing the generalizability of code2vec token embeddings," in *2019 34th IEEE/ACM International Conference on Automated Software Engineering（ASE）*.IEEE，2019年。
pp.1-12.

[38] B.Theeten, F. Vandeputte, and T. Van Cutsem, "Import2vec:Learning embeddings for software libraries," in *2019 IEEE/ACM 16th International Conference on Mining Software Repositories（MSR）*.IEEE，2019年。
pp.18-28.