

揭开源代码剽窃的面纱：利用代码结构与编程风格提高灵敏度

钟丽珊

(贵州大学 计算机科学与技术学院, 贵州 贵阳 550025)

摘要: 由于技术的进步,源代码剽窃问题日益严重。现有的代码剽窃检测方法通常只关注代码结构的相似性,往往忽视了程序员之间编程风格的差别。鉴于此,提出一种新的结合编程风格和代码结构相似性的源代码剽窃检测范式。该范式在计算代码结构相似度的基础上,同时考虑检测代码风格相似度,结合两种相似度检测结果以评判代码是否剽窃。在真实源代码剽窃和疑似源代码剽窃两种数据集上验证了所提方法的有效性。该方法不仅能提升代码剽窃检测准确度,还可提升对无意相似和有意剽窃的区分能力,为源代码剽窃检测提供了更加全面的解决方案。

关键词: 源代码剽窃;源代码剽窃检测;编程风格;源代码结构

DOI:10.11907/rjdk.241286

开放科学(资源服务)标识码(OSID):

中图分类号:TP311.1

文献标识码:A

文章编号:1672-7800(2025)005-0172-07



Unmasking Source Code Plagiarism: Leveraging Code Structure and Programming Style for Enhanced Sensitivity

ZHONG Lishan

(School of Computer Science and Technology, Guizhou University, Guiyang 550025, China)

Abstract: Source code plagiarism is an escalating issue due to technological advancements. The available detection methods disregard programming style nuances, concentrating solely on structural similarities. This paper proposes a novel source code plagiarism detection paradigm that combines programming style and code structure similarity. The paradigm, built upon calculating code structure similarity, simultaneously considers detecting code style similarity. It combines the results of both similarity detection to judge whether the code is plagiarized. Experimental validation on datasets of real source code plagiarism and suspected source code plagiarism demonstrates the effectiveness of the proposed method. This method enhances the accuracy of code plagiarism detection and improves the ability to differentiate between unintentional similarities and intentional plagiarism, which provides a more comprehensive solution for source code plagiarism detection.

Key Words: source code plagiarism; source code plagiarism detection; programming style; source code structure

0 引言

源代码剽窃是指未经允许或适当引用的情况下,擅自复制、粘贴或修改他人的程序代码,并将其用于自己的项目或作品中,而不注明原作者或未遵循适当的授权和许可。随着科技的迅速发展,源代码剽窃现象变得愈发严重。因此,源代码剽窃问题是一个亟需关注的严峻挑战。目前,许多用于检测剽窃问题的方法被提出,并取得了显著效果。最新流行的方法是基于 Running Karp-Rabin

Greedy String Tiling(RKR-GST)的结构相似性检测方法,该方法拥有强大的字符串匹配能力,能够同时使用多个模式检测剽窃代码的结构相似性,并有效地发现源代码中的剽窃行为^[1]。

然而,该检测方法忽略了源代码剽窃检测中编程风格的重要作用。编程风格是指程序员在编写代码时形成的独特风格和惯用写法,包括变量命名、代码结构、注释规范等,编程风格具有个性化的特质,剽窃者难以完全模仿原作者的编码风格。深入挖掘这些细微但独特的编程风格特征,源代码剽窃检测系统能更准确地区分无意剽窃和蓄

收稿日期:2024-04-23

扫描二维码阅读全文:

作者简介:钟丽珊(1997-),女,贵州大学计算机科学与技术学院硕士研究生,研究方向为文本挖掘、文本聚类。



意剽窃。剽窃检测时忽略编程风格会导致检测准确性存在一定局限。为了解决该问题,本文提出一种联合检测机制,在检测是否剽窃时同时考虑编程风格相似性和代码结构相似性。该方法在RKR-GST对代码结构相似检测的基础上,同时比较源代码的编程风格特征以发现潜在的剽窃行为。这一方法有助于排除偶然的相似性,在使用代码结构相似性检测剽窃的基础上进一步提高剽窃检测准确性。

本文提出的方法是一种相对全面的源代码剽窃检测方案,通过引入代码中隐藏的编程风格,提高了检测准确性和鲁棒性,解决了现有检测方法中存在的局限性。通过在真实源代码剽窃和疑似源代码剽窃这两种数据集上进行实验,验证了所提方法的有效性。该方法在提升代码剽窃检测准确度的同时,也提升了对无意相似和有意剽窃的区分能力,是加强源代码剽窃检测的有效途径。

1 相关工作

编程作为社会信息化转型的关键技术,已渗透到人们生活的方方面面,成为推动社会进步和发展的重要力量^[2]。然而,随着互联网的迅速发展,源代码剽窃成为一个长期存在的问题。代码剽窃检测作为软件工程和信息安全领域的一个重要研究方向,近年来受到了广泛关注^[3-4]。此类源代码检查技术可分为基于属性计数和基于结构两种方法。基于属性计数的方法根据源代码中存在的属性计算源代码相似度,而基于结构的方法则根据源代码的顺序/结构评估源代码相似性。相比属性计数,基于结构的方法更加有效,例如字符串匹配算法^[5-6]。一些常用的源代码剽窃检测工具也在其方法中应用了字符串匹配算法,如JPlag^[7]、YAP3^[8]、Plagie^[9]、Deimos^[10]、ES-Plag^[11]、Sim^[12]和Sherlock^[13]。

另外,利用所有权分析进行源代码剽窃检测的方法也受到了关注,所有权分析的目的是识别源代码的作者^[14]。其中,有些技术依赖于SCAP方法,该方法使用N-Grams测量代码的相似性,并对程序及其作者进行分类。文献[15-16]均用到了N-Grams方法,特别是在文献[16]中,N-Grams被用作查询文档表示法,且SCAP被用来比较每个作者简介。如上所述,SCAP方法与N-Grams密切相关。通过搜索引擎的信息检索获得N-Grams索引,之后利用N-Grams索引可以确定程序的作者^[17]。另一种利用信息检索的技术见文献[18]。文献[19]在逻辑模型上使用N-Grams构建程序员的作者档案。此外,部分研究中利用了机器学习。具体而言,有基于聚类技术和基于分类的技术。基于聚类技术在文献[20]中采用了模糊C均值算法,在文献[21]中采用了PAM,以及在文献[22]中采用了特征聚类。同时,基于分类的技术采用了支持向量机^[23]、随机森林和梯度提升^[24]。尽管以上工作在源代码剽窃检测问题上取得了良好成效,但并未深入探索源代码作者的

编程风格(命名风格、代码布局、注释风格等),因为在现实应用场景中,不同的源代码作者通常有独特的编码习惯。因此,在源代码剽窃检测工作中,考虑源代码作者的编程风格是一个非常值得关注的课题。

2 源代码剽窃检测设计

2.1 代码与风格分析

字符串匹配是当前源代码剽窃检测中的一种常用算法,是一种基于代码结构的方法。与基于属性计数的方法相比,基于结构的方法被部分学者认为在源代码剽窃检测中更加有效,然而仅考虑代码结构的相似性存在一定局限^[25]。具体而言,基于结构的方法侧重于代码的排列和语法,忽略了编码习惯和文体选择之间的差异。因此,在检测源代码剽窃时,编程风格相似性也是一个重要参考因素。编程风格即个人的编程习惯和偏好,通过识别每个程序员特有的风格以丰富源代码剽窃检测过程,可以进行更深入的代码分析并改善仅考虑结构相似性而存在的局限。

2.2 检测流程

基于上述分析,本文提出了一种融合代码结构分析和编程风格评估的源代码剽窃检测方法。首先输入源代码并执行源代码标记化操作,即基于ANTLR将源代码分解为中间形式的表示,即标记^[26]。因为标记对于获取源代码的语法结构和特征信息非常有用,所以这一操作也是源代码剽窃检测流程的基本操作。此外,源代码还将执行组合配对操作,即以组合方式生成源代码对用于后续相似性检测。为衡量所有代码之间相似性生成的源代码对集合 M ,集合大小为 C_n^2 ,其中 n 为源代码项目数。

上述标记化过程生成的每个标记都包含两个基本元素:属性和其对应的值。属性指用于识别标记类型的语法元素,如关键字、标识符、字面量和运算符等。值则代表从中间形式中提取的字符串字面量。上述生成的标记会执行源代码对比操作,其中包括两个子流程:基于代码结构的比较和基于编程风格的比较。本文以任意源代码对 $\{(A, B) | (A, B) \in M\}$ 为例进行源代码剽窃检测流程的详细阐述,其检测流程如图1所示。

(1)基于代码结构的比较。即利用RKR-GST算法评估代码之间的结构相似性^[1]。具体而言,基于RKR-GST算法对由标记的值组成的标记序列进行字符串匹配。给定任意代码对 (A, B) 的标记序列 S_A, S_B ,二者之间的相似度由具有惩罚机制的平均归一化确定^[11]。具体计算过程如式(1)所示。

$$s_{sim}(S_A, S_B) = \frac{2 \times (\text{matched}(S_A, S_B) - (P - 1))}{|S_A| + |S_B|} \quad (1)$$

其中, $\text{matched}(S_A, S_B)$ 表示标记序列 S_A, S_B 之间匹配的标记数, P 表示惩罚值,其中体值由匹配子序列的数量确定。

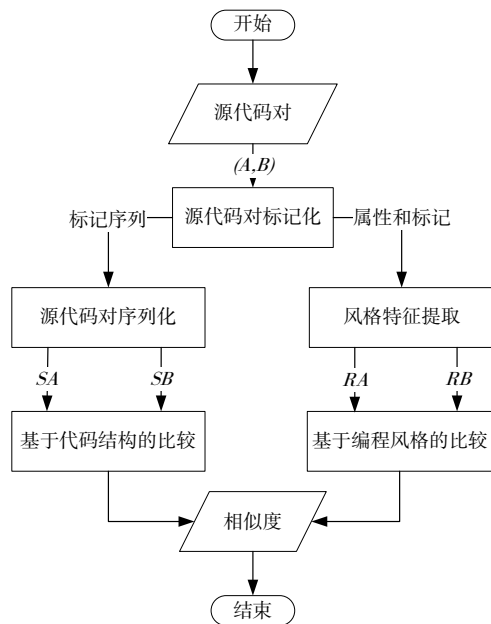


Fig. 1 Source code plagiarism detection process based on structure and programming style similarity

图1 基于结构与编程风格相似性的源代码剽窃检测流程

(2)基于编程风格的比较。即利用标记属性和属性值提取编程风格特征并进行编程风格评估。首先,本文根据实际代码编写过程,总结并提取了代表程序员风格的特征,具体风格测量属性如表1所示。

Table 1 Programming style features
表1 编程风格特征

ID	描述
PSI1	缩进中的制表符
PSI2	缩进中的空格
PSI3	空格占左侧运算符的百分比
PSI4	空格占右侧运算符的百分比
PSI5	括号和条件语句和循环语句在同一行的百分比
PSI6	空行占代码总行的百分比(不包括注释和空行)
PSI7	每行长度的平均字符数
PSI8	命名类时的常用词
PSI9	方法命名中的常用字
PSI10	命名可选代变量时的常见字符或单词
PSI11	变量命名中的常用字
PSI12	类命名中的大小写样式
PSI13	函数命名中的大小写样式
PSI14	变量命名中的大小写风格
PSI15	条件语句中"if"的百分比
PSI16	条件语句中"else if"的百分比
PSI17	条件语句中"else"的百分比
PSI18	循环语句中使用"for"的百分比
PSI19	循环语句中使用"while"的百分比
PSI20	代码行包含"import"的百分比
PSI21	源代码中的类总数
PSI22	源代码中的函数总数

提取的编程风格特征包括数字特征和非数字特征两种。因此,给定任意代码对 (A, B) 的编程风格特征表示 R_A, R_B ,本文采用了两种不同的相似度计算方法:欧式距离和杰卡德相似系数。具体操作过程如式(2)、式(3)所示。

$$sty_{sim}(R_A, R_B) = \sqrt{\sum_{i=1}^k (a_i - b_i)^2} \quad (2)$$

$$sty_{sim}(R_A, R_B) = \frac{|R_A \cap R_B|}{|R_A \cup R_B|} \quad (3)$$

其中,式(2)用于计算数字特征之间的相似度, a_i 和 b_i 分别表示 R_A 和 R_B 在第 i 维的特征值, k 表示编程风格特征个数。式(3)用于计算非数字特征之间的相似度,即将 R_A 和 R_B 的交集进行联合计算。

(3)总体相似度。即通过基于代码结构的比较和基于编程风格的比较以计算任意代码对 (A, B) 之间的相似度。具体计算过程如式(4)所示。

$$ssty_{sim}(A, B) = 0.9 \times s_{sim}(S_A, S_B) + 0.1 \times sty_{sim}(R_A, R_B) \quad (4)$$

其中, $s_{sim}(S_A, S_B)$ 表示结构相似度, $sty_{sim}(R_A, R_B)$ 表示编程风格相似度。 $s_{sim}(S_A, S_B)$ 的权重大于 $sty_{sim}(R_A, R_B)$ 的权重,原因在于结构相似性是检测源代码剽窃的主要因素^[27],而编程风格相似性则可以更好地挖掘编程习惯,有助于源代码剽窃的检测。

最后,为了衡量所有代码之间的相似性,本文依次遍历集合 M 中的 C_n^2 个源代码对并执行上述两种比较操作,根据所有代码对的总体相似度之和进行阈值判断,最终确定是否存在源代码剽窃情况。通常情况下,本文将阈值设置为50%,即假设相似度超过50%就表示存在源代码剽窃。

3 实验与结果分析

3.1 数据集

本文实验所用的代码剽窃数据集主要分为两类:一是真实源代码剽窃数据集^[28],二是疑似源代码剽窃数据集。在这两类数据集上设置了两组实验,以验证所提方法的有效性。数据集详细描述如表2所示。

3.1.1 真实源代码剽窃数据集

真实源代码剽窃是一类具有真实世界剽窃场景的数据集,即能确切地知悉数据集中的代码是否为剽窃。本文实验中采用两个真实源代码剽窃数据集进行实验:一个是根据不同的编程问题进行构造,总计有7个问题,如表3所示;另一个是根据剽窃级别划分构造,该数据集根据剽窃的程度从低到高总共有6个级别,如表4所示。这两个数据集均由9名程序员扮演剽窃者,模拟真实剽窃场景生成,其中对比对数量指源代码样本之间执行相似检测的数量。

3.1.2 疑似源代码剽窃数据集

疑似源代码剽窃数据集由程序设计入门课程和算法与结构数据两门课程学生作业的源代码整合而成。其中,程序设计入门课程的源代码来源于21名学生,算法和结构数据课程源代码来源于14名学生。本文从上述数据集中挑选相似度大于50%的源代码作为疑似剽窃的源代码,构造疑似源代码剽窃的数据集。

Table 2 Dataset scenario

表 2 数据集场景

类别	数据集	程序员数量	对比对数量
真实源代码剽窃	基于问题	9	356
	基于剽窃级别	9	356
疑似源代码剽窃	程序设计入门课程	21	2 724
	算法和结构数据课程	14	1 283
竞争性	竞争性数据集	20	-
	竞争性—无意相似数据集	5	1 225

Table 3 Programming problems of real plagiarism dataset

表 3 真实剽窃数据集的编程问题

ID	问题
C01	程序输出
C02	程序输入输出
C03	条件语句
C04	循环语句
C05	程序函数
C06	数组
C07	二维数组

Table 4 Real source code plagiarism data set based on source code plagiarism level taxonomy

表 4 基于源代码剽窃级别分类法的真实源代码剽窃数据集

ID	剽窃
L1	注释和缩进的更改
L2	级别 1 的变化和标识符的变化
L3	级别 2 中的更改和声明中的更改
L4	第 3 级的变化和程序模块的变化
L5	级别 4 的更改和程序语句的更改
L6	级别 5 的变化和决策逻辑的变化

3.1.3 竞争性数据集

竞争性数据集是从 2020 年的 Google Code Jam (GCJ) 竞赛编程中获取的数据集。该数据集分为两部分:竞争性数据集和竞争性—非故意相似数据集。竞争性数据集包括 20 个程序员,每个程序员解决 30 个编程问题。与此同时,竞争性—无意相似数据集用于分析意外相似代码的相似性分布情况。

3.2 实验评测指标

实验中使用的评测指标如下:

(1)精确度。精确度衡量的是在所有被检测为剽窃的样本中,有多少是真正的剽窃。其取值范围为(0,1),精确度越高表示模型能够在判断为剽窃的样本中较少地产生误报。计算如式(5)所示。

$$precision(i) = \frac{|relevant_retrieved(i)|}{|retrieved_documents(i)|} \quad (5)$$

(2)mAP。mAP(平均精度均值)是剽窃检测中一项重要的性能评估指标。其取值区间为(0,1),mAP 值越接近于 1 表示检测性能越好。计算公式如式(6)所示。

$$mAP(Q) = \frac{\sum_{q \in Q} AP(q)}{|Q|} \quad (6)$$

其中, $AP(q)$ 计算公式如式(7)所示。

$$AP(q) = \frac{\sum_{i=1}^n precision(i) \times is_rel(i)}{|R|} \quad (7)$$

其中, $|Q|$ 是查询总数,查询是指源代码剽窃的目标, AP 是平均精度, $precision$ 为精确度, is_rel 指第*i*个结果的真实相关性分数, $|R|$ 是给定查询的相关文档数, $relevant_retrieved$ 表示检测出的剽窃数量, $retrieved_documents$ 表示实验中用于剽窃检测的源代码总数量。

3.3 实验设计与结果分析

3.3.1 编程风格特征分析重要性

实验的主要目标是评估编程风格特征在区分源代码不同程序员方面的重要性。通过方差分析(ANOVA)测试,旨在确定在多个程序员之间的编程风格特征均值是否存在统计上的显著差异。这项调查对于了解编程风格属性能否作为个人编码习惯和作者身份的可靠标志至关重要。

为执行实验,本文汇编了一个包含竞争性数据集的源代码数据集,每个样本都标注有编程风格特征。接着,根据程序员对数据集进行分组,并进行方差分析以比较这些组之间的编程风格特征均值。该实验结果如表 5 所示。

假设: H_0 表示不同程序员之间的编程风格特征均值没有显著差异; H_1 表示不同程序员之间的编程风格特征均值存在显著差异。

Table 5 F-value and p-value analysis

表 5 F 值与 p 值分析

特征	F 值	p 值	显著性($\alpha=.05$)
PSI1	523.875	<.001	拒绝原假设
PSI2	59.648	<.001	拒绝原假设
PSI3	150.019	<.001	拒绝原假设
PSI4	140.598	<.001	拒绝原假设
PSI5	73.306	<.001	拒绝原假设
PSI6	65.791	<.001	拒绝原假设
PSI7	138.758	<.001	拒绝原假设
PSI8	27.377	<.001	拒绝原假设
PSI9	20.688	<.001	拒绝原假设
PSI10	31.142	<.001	拒绝原假设
PSI11	8.749	<.001	拒绝原假设
PSI12	28.218	<.001	拒绝原假设
PSI13	12.729	<.001	拒绝原假设
PSI14	1.634	0.044	拒绝原假设
PSI15	8.976	<.001	拒绝原假设
PSI16	5.664	<.001	拒绝原假设
PSI17	22.775	<.001	拒绝原假设
PSI18	50.532	<.001	拒绝原假设
PSI19	116.172	<.001	拒绝原假设
PSI20	316.124	<.001	拒绝原假设
PSI21	85.209	<.001	拒绝原假设
PSI22	233.32	<.001	拒绝原假设

p 值是针对零假设的证据度量。而 F 值则是组间方差与组内方差的比率。实验结果显示,22 个特征中有 21 个特征的 p 值小于 0.001 ($p<.001$),还有一个特征的 p 值为

0.044。这表明组间观察到的差异具有统计学意义。换言之,有强有力的证据拒绝零假设,表明至少有一个组的均值与其他组不同。此外,高F值意味着组间均值的变异性远大于每个组内的变异性,这支持了存在显著差异的结论。总体而言,所提出的特征对响应变量有较强影响,而各组的均值很可能存在显著差异。

3.3.2 编程风格特征有效性验证实验

该实验旨在评估编程风格特征在利用分类模型区分不同程序员方面的有效性,采用了各种模型,包括随机森林(RF)、多项式朴素贝叶斯(MNB)、支持向量分类器(SVC)和梯度提升机(GBM)以执行分类任务。实验中利用了竞争性数据集。

实验中对分类模型应用了10折交叉验证。这种方法涉及将数据集分成10个大小相等的折叠,其中每个折叠作为一个测试集,而其余数据作为训练集。通过汇总所有10个折叠的结果以计算最终性能指标,包括精确度、召回率和F-Score。

Table 6 Effectiveness metrics of the competitive dataset

表6 竞争性数据集的有效性指标

模型	精确	召回	F-Score
Random Forest	0.956 67	0.955	0.954 13
Multinomial Naïve Bayes	0.766 238	0.751 667	0.747 536
Support Vector Machines	0.795 566	0.77	0.763 511
Gradient Boosting Machines	0.914 157	0.91	0.910 347

该实验旨在评估每个分类模型如何利用编程风格特征准确地对代码样本进行分类并识别潜在的程序员。通过比较不同模型在这些指标上的表现,深入了解编程风格特征对于作者归因任务的适用性。此外,还旨在确定哪种分类模型在利用这些特征进行准确分类方面表现最佳。根据表6结果可知,随机森林分类在其他模型中表现出色。图2描绘了模型对20位程序员能力的准确分类,结果显示SVC和MNB模型对P3、P6、P7、P10和P15进行分类的准确率较低,而RF和GBM模型能够准确对程序员加以分类。

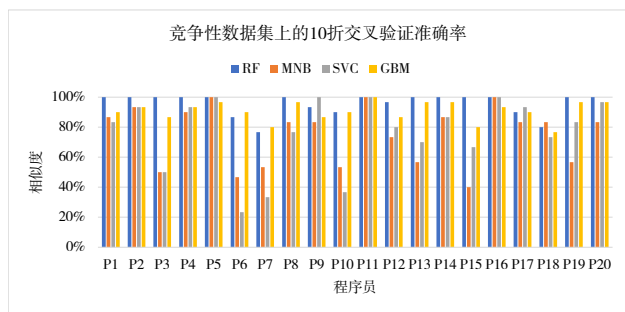


Fig. 2 10-fold cross-validation accuracy on competitive dataset

图2 竞争性数据集上的10折交叉验证准确率

3.3.3 模型有效性与编程风格相似性对提高剽窃检测灵敏度的验证实验

该实验为了评估所提方法在检测源代码是否为剽窃的性能,使用本文设计的两种相似性计算方法对剽窃代码

与原始代码的平均精确度(mAP)进行了比较。一种是基于结构的相似性检测机制 s_{sim} ,一种是整合结构相似性评估和编码风格相似性集成评测机制 $ssty_{sim}$ 。后者在评测源代码是否为剽窃时综合考虑了结构与编程风格相似性,这也是本文设计的最优评测机制。

该实验在真实源代码剽窃数据集上进行验证,通过对比 $ssty_{sim}$ 和 s_{sim} ,验证了本文 $ssty_{sim}$ 的有效性。此外,该实验相当于对 sty_{sim} (基于编程风格的相似性检测)进行了消融,也验证了结合编程风格相似性进行评测剽窃问题的有效性。

实验结果如图3和图4所示。

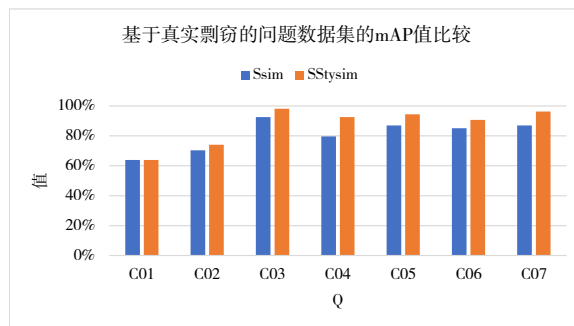


Fig. 3 Comparison of mAP values based on programming cases dataset

图3 基于编程案例数据集的mAP值比较

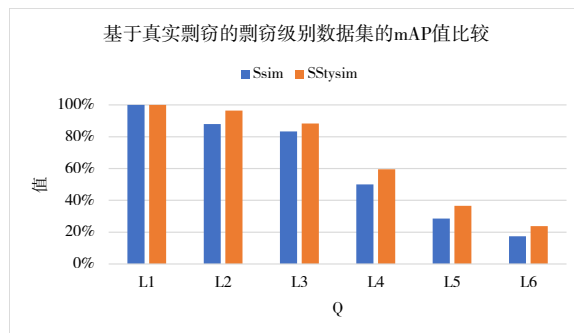


Fig. 4 Comparison of mAP values based on plagiarism level dataset

图4 基于剽窃级别数据集的mAP值比较

实验证明,本文方法(基于 $ssty_{sim}$ 进行评测)在剽窃级别的情境下能够有效地检测出源代码是否存在剽窃。这也表明,通过结合基于代码结构的比较和基于编程风格的比较,有效提升了检测源代码蓄意剽窃的性能。

3.3.4 程序设计风格相似性对检测源代码剽窃的有效性分析

为进一步探讨检测源代码剽窃时考虑编程风格的合理性,本文就编程风格对代码剽窃检测的影响在疑似源代码剽窃数据集(编程入门课程数据集与算法与数据结构课程数据集)和真实源代码剽窃数据集上关于相似度分布进行验证。在疑似源代码剽窃数据集上的实验筛选出 ord_{sim} 相似度超过50%的源代码对以进行实验。 ord_{sim} 是本实验中使用的一种基线技术,它是一种常见技术,将代码转换为标记,并使用RKGST进行比较,其最小匹配长度

为两个。验证基于结构相似性 ord_{sim} 检测出的高剽窃数据同时在 $ssty_{sim}$ 评测标准上反映突出。剽窃实验结果如图5和图6所示。其中,长方形区域表示程序设计入门可疑配对数据集和算法与结构数据可疑配对数据集的相似性程度分布空间,长方形的上界表示这批源代码按照对应评测指标估算的最高相似度,下界表示这批源代码按照对应评测指标估算的最低相似度。

对两个可疑配对数据集相似性分布的分析结果显示,经过 $ssty_{sim}$ 测算的相似度分布较 ord_{sim} 均有所上升,第一个数据集的平均相似度下降了7.14%,第二个下降了7.54%。这表明本文提出的技术更容易被修改,因为它考虑了风格相似性,而不仅仅是依赖内容。检测剽窃过程中考虑代码风格可进一步提升代码剽窃检测的可靠性,这说明在剽窃检测过程中考虑编程风格相似性合理且必要。

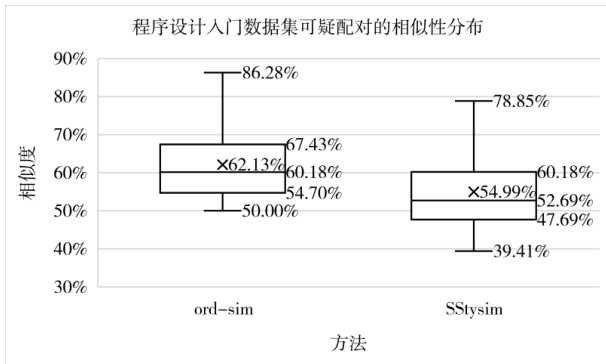


Fig. 5 Similarity distribution of suspected source code plagiarism pairs in introduction to programming subject
图5 程序设计入门可疑配对数据集相似性分布

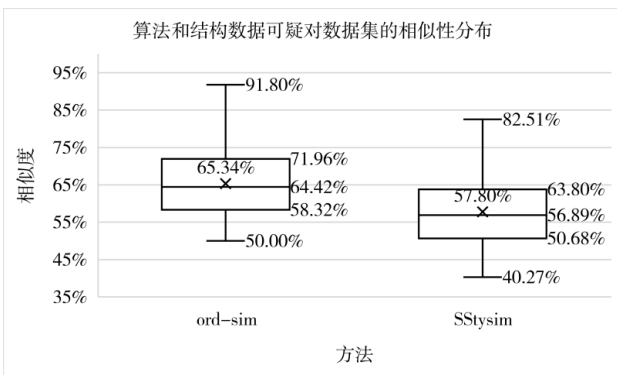


Fig. 6 Similarity distribution of suspected source code plagiarism pairs in algorithm and data structure subject
图6 算法与结构数据可疑配对数据集相似性分布

然而,值得注意的是,高相似度的源代码可能源于开发者间的无意识相似而非有意剽窃。比如,两个独立工作的开发者可能在解决同一问题时采用类似的实现方式,仅凭源代码结构进行比对,会呈现出高度相似性。这种情况在学生的课堂作业中尤为常见,因为他们在形成自己独特编程风格的过程中,可能会无意间产生相似代码。因此,在源代码剽窃检测过程中,综合考虑编程风格至关重要,因为通常情况下,独立的开发者会拥有各自独特的编程风

格。为了进一步验证这一点,本文利用真实的源代码剽窃案例和竞赛性数据集进行了深入实验。在这些实验中,重点关注那些经过确认的非剽窃代码,并在其基础上进行验证,实验结果如图7所示。分析无意相似性代码可知,在仅依赖结构相似性 ord_{sim} 进行判定的情况下,高相似度的源代码在结合编程风格进行综合考虑后,其平均相似度下降了8.3%。此外,在竞赛性数据集中,已确认不存在任何作弊和抄袭行为,故那些高相似度的代码对可被视为非巧合相似。实验结果同样如图8所示,结果显示平均相似度下降了10.12%。该结果表明,在剽窃检测中,充分考虑编程风格能够有效缓解因无意相似带来的误判问题,并显著提升判断源代码是否涉及剽窃的准确性。

综上所述,以上实验结果均验证了考虑编程风格以评测是否剽窃是合理且有效的。

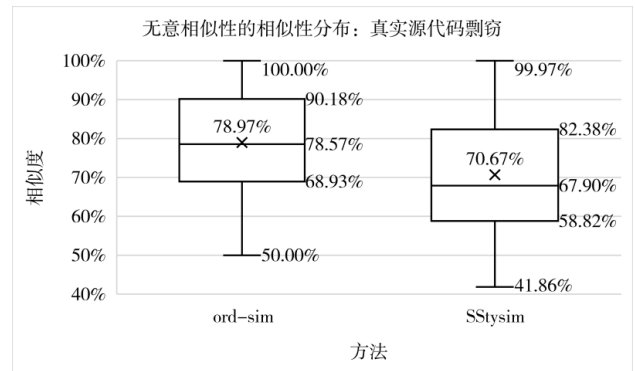


Fig. 7 Similarity distribution of unintentional similar source code pairs observed on real source code plagiarism dataset
图7 在真实源代码剽窃数据集上无意相似性相似性分布

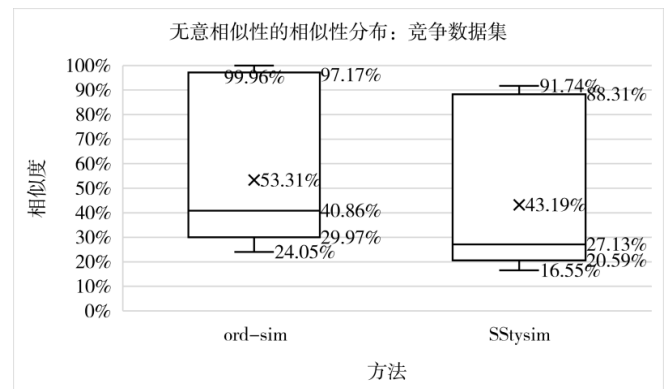


Fig. 8 Similarity distribution of unintentional similarity observed on the competitive dataset
图8 在竞赛数据集上的无意相似性相似性分布

4 结语

本文提出了一种新颖的代码剽窃检测方法,该方法能够同时考虑代码的结构相似性和编程风格相似性。通过RKR-GST算法计算代码结构相似度,并在此基础上引入代码编程风格相似性,综合两种相似性进行代码分析,从而提高了代码剽窃的检测能力。相比仅使用结构相似度

的方法,结合编程风格的方法具备更高的检测性能,证明了本文方法的有效性。为了解决如何区分无意相似和故意剽窃的问题,本文在真实的剽窃数据集上进行了额外实验。实验结果表明,本文方法对代码的无意相似具有很强的识别能力。未来研究中,将探索更加丰富的编程风格特征,进一步提高代码剽窃检测准确性。

参考文献:

- [1] WISE M J. Running Karp-Rabin matching and greedy string tiling[M]. Sydney: University of Sydney, 1993.
- [2] YUE K, HU K, YUAN G W, et al. Establishment of online program design platform and practice education reform under open-source architecture[J]. Software Guide, 2023, 22(3): 212-216.
岳昆, 胡矿, 袁国武, 等. 开源框架下在线编程平台建设与实践教学改革[J]. 软件导刊, 2023, 22(3): 212-216.
- [3] LI G F, ZHANG F, LIU C. CLPDetector: a cross-language code plagiarism detection tool based on pseudo-siamese network[J]. Journal of Chinese Computer System, 2022, 43(7): 1562-1568.
李国繁, 张峰, 刘聪. CLPDetector: 一种基于伪孪生网络的跨语言代码抄袭检测工具[J]. 小型微型计算机系统, 2022, 43(7): 1562-1568.
- [4] ZHANG F, WEI Y L, QIN Y C. Cross language code plagiarism detection based on program flow chart and graph attention[J]. Journal of Chinese Computer System, 2025, 46(1): 249-256.
张峰, 韦友良, 秦玉成. 基于程序流程图和图注意力网络的跨语言代码抄袭检测方法[J]. 小型微型计算机系统, 2025, 46(1): 249-256.
- [5] WHALE G. Identification of program similarity in large populations[J]. The Computer Journal, 1990, 33(2): 140-146.
- [6] WHALE G. Software metrics and plagiarism detection[J]. Journal of Systems and Software, 1990, 13(2): 131-138.
- [7] PRECHELT L, MALPOHL G, PHILIPPSEN M, et al. Finding plagiarisms among a set of programs with JPlag[J]. Journal of Universal Computer Science, 2002, 8(11): 1016.
- [8] WISE M J. YAP3: improved detection of similarities in computer program and other texts[C]//Proceedings of the Twenty-seventh SIGCSE Technical Symposium on Computer Science Education, 1996: 130-134.
- [9] AHTIAINEN A, SURAKKA S, RAHIKAINEN M. Plaggie: GNU-licensed source code plagiarism detection engine for Java exercises[C]//Proceedings of the 6th Baltic Sea Conference on Computing Education Research, 2006: 141-142.
- [10] KUSTANTO C, LIEM I. Automatic source code plagiarism detection[C]//2009 10th ACIS International Conference on Software Engineering, Artificial Intelligences, Networking and Parallel/Distributed Computing, 2009: 481-486.
- [11] SULISTIANI L, KARNALIM O. ES-Plag: efficient and sensitive source code plagiarism detection tool for academic environment[J]. Computer Applications in Engineering Education, 2019, 27(1): 166-182.
- [12] GITCHELL D, TRAN N. Sim: a utility for detecting similarity in computer programs[J]. ACM Sigcse Bulletin, 1999, 31(1): 266-270.
- [13] JOY M, LUCK M. Plagiarism in programming assignments[J]. IEEE Transactions on Education, 1999, 42(2): 129-133.
- [14] FRANTZESKOU G, MACDONELL S, STAMATATOS E, et al. Examining the significance of high-level programming features in source code author classification[J]. Journal of Systems and Software, 2008, 81(3): 447-460.
- [15] TENNYSON M F. A replicated comparative study of source code authorship attribution[C]//Proceedings of 2013 3rd International Workshop on Replication in Empirical Software Engineering Research, 2013: 76-83.
- [16] TENNYSON M F, MITRPOULOS F J. Choosing a profile length in the SCAP method of source code authorship attribution[C]//2014 Proceedings of the IEEE SoutheastCon, 2014: 1-6.
- [17] BURROWS S, TAHAGHOGHI S M M. Source code authorship attribution using N-Grams[C]//Melbourne: the 12th Australasian Document Computing Symposium, 2007.
- [18] BURROWS S, UITDENBOGERD A L, TURPIN A. Application of information retrieval techniques for source code authorship attribution[C]//DASFAA 2009: Database Systems for Advanced Applications, 2009: 699-713.
- [19] ZHANG C, WANG S, WU J, et al. Authorship identification of source codes[C]//Asia-Pacific Web (APWeb) and Web-Age Information Management (WAIM) Joint Conference on Web and Big Data, 2017: 282-296.
- [20] ACAMPORA G, COSMA G. A fuzzy-based approach to programming language independent source-code plagiarism detection[C]//Istanbul: 2015 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE), 2015.
- [21] OHMANN T, RAHAL I. Efficient clustering-based source code plagiarism detection using PIY[J]. Knowledge and Information Systems, 2015, 43(2): 445-472.
- [22] YASASWI J, KAILASH S, CHILUPURI A, et al. Unsupervised learning based approach for plagiarism detection in programming assignments[C]//Proceedings of the 10th Innovations in Software Engineering Conference, 2017: 117-121.
- [23] YASASWI J, PURINI S, JAWAHAR C V. Plagiarism detection in programming assignments using deep features[C]//Proceedings of 4th Asian Conference on Pattern Recognition, 2018: 623-628.
- [24] HUANG Q B, TANG J D, FANG G Z. Research on code plagiarism detection model based on random forest and gradient boosting decision tree[C]//Proceedings of the 2019 International Conference on Data Mining and Machine Learning, 2019: 97-102.
- [25] KARNALIM O, SULISTIANI L. Which source code plagiarism detection approach is more humane?[DB/OL]. <http://arxiv.org/abs/1809.08559>, 2018.
- [26] PARR T. The definitive ANTLR 4 reference[M]. Raleigh: Pragmatic Bookshelf, 2013.
- [27] KARNALIM O, KURNIAWATI G. Programming style on source code plagiarism and collusion detection[J]. International Journal of Computing, 2020, 19(1): 27-38.
- [28] KARNALIM O, BUDI S, TOBA H, et al. Source code plagiarism detection in academia with information retrieval: dataset and the observation[J]. Informatics in Education, 2019, 18(2): 321-344.

(责任编辑:孙娟)