一种基于BP神经网络的代码相似性检测方法

熊 浩[1,2] 晏海华[1] 黄永刚[2] 郭 涛[2] 李舟军[1]
(北京航空航天大学计算机学院 北京 100191)[1] (中国信息安全评测中心 北京 100083)[2]

摘 要 如何有效地检测程序设计课程作业中的抄袭现象是一个重要的问题。传统的抄袭检测方法主要利用代码的属性或结构信息来度量代码之间的相似性。给出了一种基于误差反向传播(BP算法)多层前向神经网络的代码抄袭检测方法。提取程序之间的7种比较特征作为神经网络的输入,经过网络计算后得出程序的相似值,并将该值与抄袭决策阈值相比较以判定存在抄袭现象的程序集。实验结果表明,本方法具有很好的检测效果。

关键词 抄袭,相似性检测,BP神经网络,比较特征
中图法分类号 TP311 **文献标识码** A

Code Similarity Detection Approach Based on Back-propagation Neural Network

XIONG Hao[1,2] YAN Hai-hua[1] HUANG Yong-gang[2] GUO Tao[2] LI Zhou-jun[1]
(School of Computer Science and Engineering, Beihang University, Beijing 100191, China)[1]
(China Information Technology Security Evaluation Center, Beijing 100083, China)[2]

Abstract It is very important to find plagiarized programs in the field of computer science education. Traditional methods for program similarity use attribute counting or structure information to detect plagiarism. This paper presented a program similarity detection approach based on back propagation (BP algorithm) multi-layer feed-forward neural networks. We extracted seven compared features of the code as the input of the neural network, and obtained the program similarity through the network calculation. Comparing the result with the threshold value, we can find all groups of similar programs. Experimental results show that our method is effective.

Keywords Plagiarism, Similarity detection, BP neural network, Compared feature

当前国内外大学教育中抄袭现象非常普遍。特别是程序设计课程,作业抄袭现象尤为严重[1]。有调查报告显示,大部分的学生承认抄袭过别人的编程作业[2,3]。这与编程作业这类电子文件易于拷贝的特点有很大的关系。相比自然语言,程序语言的语法规则比较简单,抄袭者只需要通过编辑器改变代码外观就可达到抄袭目的。

传统的人工抄袭检测方法耗时费力,而且代码规模和教师的经验对检测结果有很大的影响。如今已经有很多的软件工具辅助教师检测代码的相似性,以Berkeley大学的MOSS系统[9]和Karlsruhe大学的JPlag系统[10]为代表。但是,这些检测工具往往只针对某些特定的抄袭手段,检测效果并不理想。而且,经典的检测工具只能给出程序的相似值,决策程序抄袭的相似度阈值通常需要人工选定。该值的选取将直接影响抄袭判别结果,过大的阈值必然会引起误判,阈值过小也会带来漏判的现象。实际操作过程中,教师需要人工查看大量的学生程序,才能给出一个较为合适的阈值。

针对上述问题,本文提出一种基于BP神经网络的程序相似性检测方法。该方法主要分为两个阶段:其一为特征发生阶段,引入代码注释分离技术、编译技术、优化编译和反汇编技术,将代码集分解或变换成比较样本集,通过计算公式得出检测程序对的7种相似性结果,组成特征向量;其二为BP神经网络训练阶段,将比较代码的特征向量输入至BP神经网络,输出即为两代码是否抄袭的预测值。该方法的独到之处在于:引入多种技术对源代码进行变换,从程序的语义层面消除抄袭手段带来的干扰,使用神经网络整合不同技术下的相似度比较结果。与JPlag等系统对一组作业程序集进行的评测结果相比,本方法具有更好的检测效果。

1 相关研究

1.1 程序抄袭手段

Parker认为[4],如果一份代码由另一份代码完全拷贝或只做了很少量的修改得到,就可以认为此代码为抄袭代码。抄袭代码通常是花费很少的代价产生的。Faidhi和Robinson[5]归纳了7种学生常用的抄袭手段,并使用光谱来分析这些抄袭手段的代价和难易度。Jones[6]在此基础上,总结了10种代码抄袭手段。此外,还有学者在研究中提出了其他的

到稿日期:2009-04-08 返修日期:2009-07-16 本文受863国家重点基金项目(2007AA010302,2008AA012114),国家自然科学基金(60703057,60573084)资助。

熊 浩(1985—),男,硕士生,主要研究方向为数据挖掘、软件工程等,E-mail:xionhao2005@cse.buaa.edu.cn;晏海华(1964—),男,硕士,副教授,主要研究方向为软件工程、软件测试等;黄永刚 男,硕士,主要研究方向为信息安全、安全测试等;郭 涛(1974—),男,博士,主要研究方向为信息安全、安全测试等;李舟军(1963—),男,博士,教授,博士生导师,主要研究方向为进程代数理论、安全协议形式化验证、数据挖掘等。

抄袭手段[15-17]。

虽然没有权威的代码抄袭手段分类，但是抄袭者在不理解代码内部流程的情况下，要达到程序运行正确和加大检测难度这两个目的，所做的代码变化一般集中于3个方面：

1)代码布局：比如重新排版、代码段的重排序等。

2)语法理解：比如标识符换名、等价关键字的替换等。

3)语义分析：比如函数分解与结合、等价语句块变换等。

一般来说，代码在1)、2)两个方面的变化所付出的代价很小，只需要使用编辑器改变代码的外观，Faidhi归纳的7种和Jones总结的10种抄袭手段均集中在这两个方面。抄袭手段在语义层面需要付出较高的代价，但语义分析涉及的代码语句较多，在很大程度上改变代码外观，从而加大了检测的难度。

1.2 抄袭检测技术

程序相似性检测技术的研究工作已经有40年的历史，主要有属性度量和程序结构分析两类方法。属性度量的方法是将程序视为文本，从中抽取诸如词频、语句长度等统计信息组成代码的度量元，而后使用评估公式获得程序的相似性。早期的研究工作主要集中在如何获得尽可能多的代码度量属性[4]。但是这些工作并没有很好地改善相似度比较结果。VERCO认为改进措施是考虑程序的结构信息[7]。

由于程序结构的复杂性，基于代码结构的相似性检测方法往往把源代码转换成其他形式的样本，比如语法树和标识符串，再使用算法来计算中间形态样本的相似度。表1中总结了一些比较流行的、基于结构的代码相似检测系统。

表1 代码相似检测系统

编号	系统	算法
1	YAP3[8]	RKR-GST
2	MOSS[9]	String Matching
3	JPLAG[10]	Greedy String Tiling
4	Plague[11]	LCS algorithm
5	GPLAG[12]	Graph isomorphism algorithm
6	Bandit[13]	LCS algorithm
7	SIM[14]	Local Alignment

遗憾的是，大部分的检测系统只能检测出很少量的抄袭手段。学生只需要使用一些语法和语义方面的抄袭手段，就可逃避抄袭检测。

2 基于BP神经网络的程序相似性检测方法

2.1 BP神经网络技术

人工神经网络[18]是模拟人脑生物过程的人工智能技术，由大量的神经元互连形成复杂的非线性系统。误差反向传播(BP)神经网络是目前应用最为广泛的神经网络学习算法，可以实现输入和输出间的任意非线性映射。网络由若干输入层、隐层和输出层的节点组成，如图1所示。

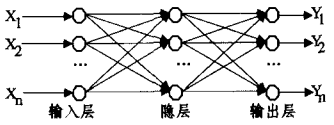

图1 BP神经网络结构

2.2 抄袭检测方法设计

基于BP神经网络的代码抄袭检测方法的核心思想是收集待检测的两份代码在不同方面的相似值，作为神经网络的输入向量；通过神经网络学习，得出是否抄袭的结论。该方法的流程如图2所示。

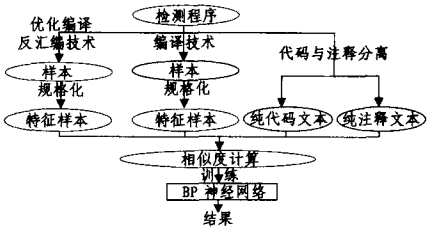

图2 基于BP神经网络的程序相似性检测方法

检测程序通过优化编译、反汇编技术和编译技术转化为样本，剔除样本中的无关信息组成特征样本，经不同计算公式得出的相似值在7个方面反映了待比较的程序之间的相似程度，并将其组成向量作为神经网络的输入，以下各小节将分别叙述这7个特征。

2.2.1 特征优化编译和反汇编

文献[17]的基本思想是，把代码通过优化编译成可执行的二进制文件，借助工具，将此二进制文件反汇编成汇编代码，从中抽取源代码程序部分的汇编指令，删除与程序无关的信息。需要指出的是，简单的调整代码布局就可以改变汇编代码中的偏移地址值和函数地址值，常量替换也能够改变立即数的值。为了消除这些噪音，在汇编代码中使用统一的标识量OFFSET，FUNCTION和CONSTANT来替换偏移地址、函数地址和立即数。

设P_1，P_2为两个待检测的程序，$F(P_1)$和$F(P_2)$表示P_1，P_2经过优化编译和反汇编后的汇编指令集，$\mathrm{Sim}(P_1,P_2)$为代码的相似度，计算公式如下。

$$\mathrm{Sim}(P_1,P_2)=(F(P_1)\cap F(P_2))/(F(P_1)\cup F(P_2)) \quad (1)$$

式(1)使用指令集交集和并集的概念，其计算结果不受指令顺序的影响，消除了代码语句顺序变动所带来的影响。但是，由于集合中不能出现重复的元素，求指令交集时会将意义不同而外观相同的指令都作为一条指令，降低了检测程序的相似值。

$$\mathrm{Sim}(P_1,P_2)=\frac{match}{match+(f(P_1)-P_1_match)+(f(P_2)-P_2_match)} \quad (2)$$

式(2)有效地解决了式(1)中存在的问题。f函数为获取汇编文本的代码行数，P_1_match为P_1汇编指令在P_2汇编指令集中匹配的行数，$match$值为P_1_match和P_2_match的最大值。

待检测程序通过优化编译和反汇编技术得到的汇编文本，经过式(1)、式(2)计算得到两个相似度值，分别为特征值一和特征值二。

2.2.2 特征编译

将代码的二进制执行文件反汇编后，失去了对原来代码程序片段的定位信息，以至于不能有效地检测一些语义分析方面的抄袭。例如学生将一段程序从原函数中抽取出来，生成一个新的函数。这种抄袭手段改变了代码的控制调用关系，则使很多检测方法失效。采用编译技术在程序的语义层还原语句的执行顺序即可识别该类抄袭。

图3为本文采用的代码编译流程。待检测程序通过编译器的词法分析、语法分析和语义分析后转化成标识符集合，标识符线性化方法消除了源代码中的函数调用关系，将执行顺

序复杂的标识符集调整为一维线性顺序的标识符流。标识符线性化方法是当出现函数调用的标识符时，将所调用的函数部分的标识符序列还原至函数调用接口处。需要强调的是，对于一般的函数直接调用关系，只需要把调用函数的标识符插入函数调用处。如果存在递归调用关系，则需要做特别的处理。直接递归调用关系（函数内部语句调用其自身函数），调用自身函数接口处的标识符统一用 CALL FUNCTION 标识符代替。间接递归调用关系（函数之间相互调用），只允许函数标识符还原一次，嵌套调用接口处统一使用 CALL FUNCTION 标识符代替。

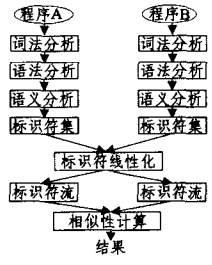

图 3　编译技术

为比较具有顺序逻辑关系的标识符序列流的相似性，首先使用动态规划算法（LCS algorithm）[19] 求出两序列的最长公共子序列。

假设需要比较的两标识符序列为 $X=\langle x_1,x_2,\cdots,x_n\rangle$ 和 $Y=\langle y_1,y_2,\cdots,y_m\rangle$，构造 $C[i,j]$ 矩阵，矩阵维数为 $m*n$，LCS 算法伪代码如下：

```
1 m=length[X],n=length[Y]
2 for i=1 to m
3     C[i,0]=0
4 for j=1 to n
5     C[0,j]=0
6 for i=1 to m
7   for j=1 to n
8       if xi=yj
9           C[i,j]=C[i-1,j-1]+1
10      else if C[i-1,j] ≥ C[i,j-1]
11          C[i,j] = C[i-1,j]
12      else C[i,j] = C[i,j-1]
13 return C
```

算法结束后，$C[m-1,n-1]$ 的值为最长公共子序列的长度，计算代码编译特征相似性的两个公式如下。

$$\mathrm{Sim}(P_1,P_2)=C[m-1][n-1]/\mathrm{min_line} \tag{3}$$

$$\mathrm{Sim}(P_1,P_2)=2*C[m-1,n-1]/(m+n) \tag{4}$$

min_line 为 m 和 n 值中的较小值。式(3)的优势在于不受冗余代码添加的抄袭手段的影响。式(4)来自信息距离[16]，其计算结果有很好的区分度。

标识符流经过式(3)、式(4)的计算得到两个相似度值，即为特征值三和特征值四。

2.2.3　属性特征提取

程序代码的属性特征，主要集中在程序风格和统计信息两个方面。而程序风格又包括代码风格和注释风格。将程序的代码段与注释行分离，分别提取这 3 个方面的特征。

虽然没有严格的风格检查标准，但我们发现学生编写程序在缩进、空白行、注释风格等几个方面较易形成个人独有的习惯。故选用 7 个代码风格特征和 4 个注释风格特征来标识程序的风格。

代码风格：

1）代码的行长字符平均值

2）代码行前缩进的空白字符平均值

3）代码行中分隔的空白字符平均值

4）代码行尾保留的空白字符平均值

5）空白代码占整个代码的百分比

6）复合语句行占整个代码行的百分比

7）左大括号位置规范出现的比例

注释风格：

1）注释的行长字符平均值

2）块注释行占整个注释行的比例

3）单行注释行占整个注释行的比例

4）单行复合注释行占整个注释行的比例

代码风格 7）中规范出现是指左大括号应该出现在声明语句的同行和复合语句起始行的行尾。注释风格 2）的块注释通常是对文件、类、方法和算法的描述，能够提高代码的可读性。块注释的标志使用/*和*/标识符，且两者不在同一行中。3）、4）的区别在于代码中单独的某一行包含的可执行的程序语句。如果不包含程序语句，则该行为单行注释行，否则为单行复合注释行。

文献[4]总结了基于代码属性度量的程序相似性检测方法，并一一列举出了各种检测方法所使用的核心代码属性，选用出现频率较高的 5 个属性作为代码的统计特征。

1）代码行数

2）赋值语句个数

3）循环控制语句个数

4）选择控制语句个数

5）自定义函数个数

赋值语句包括直接变量赋值和从终端、文本输入值。循环控制和选择控制语句以关键字为标志。比如 C 语言中，包含 for，do/while 和 while 关键字的语句为循环控制语句。而选择控制语句中的关键字有 if，switch/case，if/else 和三目运算符?:。

假设需要检测的代码集为 $P_1,P_2,\cdots,P_n$，集合中任意一份代码 P_x 的 3 类属性为：代码风格 $CS=\langle a_{1x},a_{2x},\cdots,a_{7x}\rangle$，注释风格 $RS=\langle b_{1x},b_{2x},\cdots,b_{4x}\rangle$，统计特征 $SC=\langle c_{1x},c_{2x},\cdots,c_{5x}\rangle$。采用欧式距离来度量代码集中任意两份代码 P_i 和 P_j 属性特征的相似度，以代码风格为例。

$$D(P_i,P_j)=\sqrt{\sum_{u=1}^{7}(a_{ui}-a_{uj})^2} \tag{5}$$

两属性向量的欧式距离越小，表示待检测的两份代码之间的相似度越高。为了使欧式距离值与其他特征值的值域一致（值域 0-1），需要对代码风格向量进行规格化处理。

代码风格 CS' 为

$$\langle a'_{1x},a'_{2x},\cdots,a'_{7x}\rangle=\left\langle\frac{a_{1x}}{\sum_{t=1}^{n}a_{1t}},\frac{a_{2x}}{\sum_{t=1}^{n}a_{2t}},\cdots,\frac{a_{7x}}{\sum_{t=1}^{n}a_{7t}}\right\rangle$$

如此规格化处理可以使输入式(5)的每一份代码风格的特征值在 0 与 1 的范围之内，但不能确保其计算结果在 0～1 之间，因此需要使用改进的欧式距离公式。

$$D(P_i,P_j)'=\sqrt{\sum_{u=1}^{7}(a'_{ui}-a'_{uj})^2/7} \tag{6}$$

式(6)仅从结果的角度添加一个除数，以保证公式计算结

果的范围限定于 0 到 1。待检测程序集中任意两份程序 P_i 和 P_j 的代码风格特征相似度计算公式如下。

$$\mathrm{Sim}(P_i,P_j)=1-D(P_i,P_j)' \quad (7)$$

同理，通过使用式(6)、式(7)也可以得出注释风格特征和统计特征的相似度。上述 3 个相似度值为特征值五、特征值六和特征值七。

表 2 总结了 7 个 BP 神经网络输入的比较特征，每一个特征都从不同方面反映了程序之间的相似性。通过神经网络的训练学习整和这些比较特征，以决策代码是否存在抄袭现象。

表 2 BP 神经网络输入的比较特征

名字	描述	计算公式
优化编译和反汇编代码相似性一(SCD1)	反汇编二进制文件的汇编指令集合相似度	(1)
优化编译和反汇编代码相似性二(SCD2)	反汇编二进制文件的汇编指令相似度	(2)
编译线性化标识符相似性一(SLC1)	编译后顺序标识符相似度	(3)
编译线性化标识符相似性二(SLC2)	编译后顺序标识符信息距离相似度	(4)
代码风格相似性(SCS)	待比较程序的代码风格相似度	(6)(7)
注释风格相似性(SRS)	待比较程序的注释风格相似度	(6)(7)
统计属性相似性(SSC)	待比较程序的统计属性相似度	(6)(7)

3 实验及结果评测

为了评估基于神经网络的程序相似性检测方法的效果，所做的实验包括两个部分。其一为样本训练实验，其二为应用预测实验。以某级本科生的程序设计作业作为实验的样本。分析训练样本分类效果，确定抄袭决策阈值。并且对于同一份评测样本，把预测实验的结果与其他检测系统进行比较，以发现本方法的长处与不足。实验一训练构建的 BP 神经网络是实验二的基础。

3.1 样本训练实验

3.1.1 训练数据

某高级程序设计课程的编程作业有容易、中等和较难 3 个难度等级。一般来说，容易的程序设计题目重在掌握一些简单的程序语法规则，代码之间的相似程度非常高。较难的题目通常学生的完成情况不太理想，不易获得大量的比较样本。所以，我们采集的训练数据是学生所提交的 100 份难度适中的课程作业。作业对应的问题是简易计算器的实现，程序的平均代码长度是 100 行左右，逻辑比较简单，但涉及循环和选择控制。该训练样本两两比较生成 4950 份比较样本对，经人工分析得到 0(未抄袭的样本对)和 1(抄袭的样本对)的结果。

3.1.2 实验环境

GCC 编译器是一种功能强大、性能优越的跨平台编译器。而且 Linux 系统的 Shell 脚本也提供了相应的命令行来获得 GCC 编译过程中所生成的中间代码。所以借助 GCC 编译器来收集程序编译后的相应特征。

Matlab 6.x 软件提供神经网络工具箱，为神经网络的设计和分析带来了极大的便利。通过反复实验调节，所建立的神经网络输入层和输出层分别含有 7 个和 1 个节点。而隐层有 50 个节点，该值为经验值，与实验的样本有关，可根据具体情况调节隐层的节点数。所训练的模型参数如下：训练步数为 10000；学习率为 0.05；动量常数为 0.9。

BP 神经网络在训练中通常存在局部极小和收敛性的问题，实验中的对策是调节网络中的一些参数，比如固定初始权值和偏移量、输入向量归一化以及 L-M 网络训练算法[21]等方法来降低这两个问题所带来的影响。

3.1.3 实验结果及分析

图 4 为训练效果和人工检测的预期效果比较图。X 轴为人工检测的结果，Y 轴表示 BP 神经网络的输出结果。样本经过训练得出的结果与预期结果的误差为 0.0031。这一方面是由于样本的数量很大，另一方面是部分比较样本的训练结果与预期结果出现了很大的偏差，如图 4 右下侧的几个比较样本。查看这几个对比样本的源代码后发现，抄袭程序主要是改变了代码的数据结构，即数组、链表、指针的相互变换，使得程序比较特征都下降到很低的相似水平。程序设计诸如算法思想、顺序流程等其他方面完全一样，人工检测也是根据这些方面的一致性判定为抄袭。虽然这些抄袭样本在一定程度上影响了神经网络的检测效果，但是，数据实现方式转化需要相当的编程能力，而且还需要付出很大的代价调试来确保程序运行的正确性。一般来说，教学中遇到这类抄袭的概率很小。

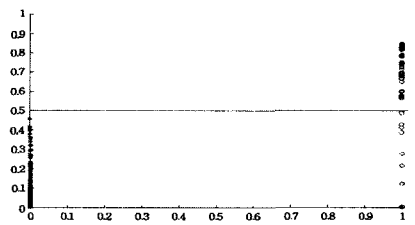

图 4 训练效果与预期效果对比

从图 4 中还可以发现，左下侧神经网络运算值小于 0.5 的比较样本聚成非抄袭类，与人工判定的非抄袭类一致。而值大于 0.5 的比较样本有 9 个比较样本被划分至非抄袭类，其余的比较样本符合人工判定的抄袭类。神经网络的运算结果以 0.5 为分界线的聚类正确率可以达到 99.818%(正确聚类的样本数/总的样本数)。故实验二中所选用的抄袭决策阈值为 0.5。

3.1.4 规一化输入特征的权值

神经元之间的连接权值很大程度上体现了输入特征对于结果的影响。神经元的权值进行数据规一化处理[20]后，得出的 7 个神经网络输入特征的影响及相应的排名如表 3 所列。各个特征都正影响于输出结果。编译特征和反汇编特征的排名比较靠前，说明这些特征能够在很大的程度上消除抄袭手段所带来的噪音，对于抄袭代码和非抄袭代码有良好的区分度。注释风格和统计特征的影响较低，表明学生的抄袭手段对代码的注释和一些统计信息有很大的影响。

表 3 输入特征的权值及排名

网络输入特征	规一化权值	排名
SCD1	0.1489	3
SCD2	0.1310	5
SLC1	0.1330	4
SLC2	0.1721	1
SCS	0.1657	2
SRS	0.1226	7
SSC	0.1267	6

3.2 应用预测实验

JPlag 是目前比较流行的相似性检测工具[10]。BuaaSim

系统对代码的抄袭现象有很好的抄袭检测效果[17]。应用预测实验是对于同一组学生提交的课程作业，将神经网络训练决策的评测结果与 BuaaSim 和 JPlag 系统的评测结果相比较，以分析神经网络方法实际中的效果。

3.2.1 预测数据

预测数据由学生提交的 75 份程序设计课程作业组成，对应的问题是输入平面上一组顶点坐标，求其围成凸多边形的面积。作业难度中等。其程序结构、所考察的知识点都与 3.1 节实验中的训练数据类似。

3.2.2 实验结果及分析

75 份程序作业(序号为 0—74)构成的 2775 份比较样本输入至神经网络中，经计算得到的结果如图 5 所示。值大于 0.5 的比较样本聚集成为抄袭类，该类中有 98 份比较样本对。其中，序号为 66 号的程序作业代码与其他 72 份程序(除 5 和 27 号程序外)生成的 72 份比较样本都在抄袭类中。这是与常理相违背的误判现象，这一现象的产生主要与实验一构建的神经网络训练误差有很大的关系，抄袭类中应该剔除这些与事实不符合的比较样本对。

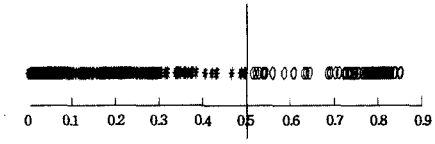

图 5 神经网络输出结果

表 4 中的检测结果是以存在抄袭的程序序号组成的抄袭集合。从比较结果可以发现，BP 神经网络方法的检测效果明显好于 BuaaSim 和 JPlag，神经网络检测出的抄袭程序中包含了这两个工具的检测结果。除此之外，还多检测出了编号为 5,6,15,17,51,67 的代码抄袭。经过人工分析，发现抄袭者对原始程序主要做了如下的修改：

1)修改声明变量的类型和数组大小；

2)从函数中抽取部分代码形成新函数；

3)与上下文有关的冗余代码的添加。

表 4 BuaaSim 和 BP 神经网络检测结果

检测方法	检测结果
JPlag	{32,45},{52,58,68}
BuaaSim	{32,45},{35,42},{56,62},{70,72},{52,58,60,68},{4,12,19}
BP 神经网络	{32,45},{35,42},{5,6},{15,16}{52,58,60,68}, {4,12,19},{15,24}{17,56,62},{67,70,72}, {24,51}{15,59},{15,63},{51,59},{51,63},{2,51}

BuaaSim 系统无法消除抄袭手段 2)所带来的影响。而且大量使用抄袭手段 1)和 3)会大大降低抄袭代码与源代码之间的相似性。此外，15 号代码和 51 号代码是由算法流程类似的其他 4 个程序拼凑形成的。抄袭程序与单个程序之间只在某些片段上存在很高的相似度，就整体而言相似度依然保持较低的水平。若相似性下降至抄袭决策阈值之下，即会使检测工具失效。

这些抄袭手段主要集中在分析程序语义方面。使用神经网络的方法，将比较程序转化为 7 个方面的相似度特征，前 4 项特征的提取过程都在一定程度上消除了语义变换带来的噪音，经过 BP 网络的计算，所得结果大于抄袭决策阈值，从而判定两份代码存在抄袭。

JPlag 遗漏的程序较多，冗余代码添加和调整函数定义位置都能很大程度影响代码之间的相似度。说明 JPlag 只能识别词法层面的抄袭。

BuaaSim 与神经网络的方法要求检测样本是可编译的，JPlag 并没有这个限制。而且，神经网络检测方法效果的好坏与训练样本有很大的关系。训练样本过小，BP 网络不具备完善的检测能力。训练样本过大，不可避免会加大误差值，从而降低分类效果。这是神经网络方法的不足之处。

结束语 通过实验分析可以看出，使用人工检测的训练样本训练并形成具有检测能力的 BP 神经网络，能有效地发现程序设计课程作业中的抄袭现象，从而保持教学质量。

下一步的工作重点将集中在以下几个方面：

- 提出其他的代码比较特征，增加神经网络输入向量的维数以期获得更好的检测效果。
- 采集并分析具有代表性的训练样本，降低神经网络训练误差从而获得更好的分类效果。
- 引入遗传算法、模拟退火及混沌搜索等算法，解决神经网络作为分类器存在局部极小、收敛性和泛化能力方面的问题。
- 将其他诸如决策树、贝叶斯网络、SVM 分类器应用于代码相似性检测，比较并分析各种分类器的优缺点。

参 考 文 献

[1] McCabe D. Levels of Cheating and Plagiarism Remain High[C/OL]. Center for Academic Integrity, Duke University, 2005. http://academicintegrity.org/

[2] Bull J, Collins C, Coughlin E, et al. Technical Review of Plagiarism Detection Software Report[C/OL]. http://www.jisc.ac.uk/pub01/luton.pdf, July 2002

[3] Sheard J, Dick M, Markham S, et al. Cheating and plagiarism: perceptions and practices of first year IT students[C]// The 7th Annual Joint Conference on Innovation and Technology in Computer Science Education. Aarhus, Denmark, 2002: 183-187

[4] Parker A, Hamblem J O. Computer algorithms for plagiarism detection[J]. IEEE Transactions on Education, 1989, 32(2): 94-99

[5] Faidhi J A W, Robinson S K. An Empirical Approach for Detection Program Similarity and Plagiarism within a University Programming Environment[J]. Computers and Education, 1987, 11(1): 11-19

[6] Jones E L. Metrics based plagiarism monitoring[C]// The 6th Annual CSSC Northeastern Conference. Middlebury, VT, 2001

[7] Verco K L, Wise M J. Software for detecting suspected plagiarism: comparing structure and attribute-counting systems[C]// Proceedings of the 1st Australian Conference on Computer Science Education. 1996: 3-5

[8] WISE M J. YAP3: Improved Detection of similarities in computer program and other Texts[C]// ACM SIGCSE. 1996: 130-134

[9] Schleimer S, Wilkerson D, Aiken A. Winnowing: Local Algorithms for Document Fingerprinting[C]// Proceedings of the ACM SIGMOD International Conference on Management of Data. June 2003: 76-85

[10] Prechelt L, Malpohl G, Philippsen M. Finding plagiarisms among a set of programs with JPlag [J]. Journal of Universal Computer Science, 2002, 8(11): 1016-1038

[11] Whale G. Plague: Plagiarism Detection Using Program Structure

[R]. Dept. of Computer Science Technical Report 8805. University of NSW, Kensington, Australasian, 1988

[12] Liu Chao, Chen Chen, Han Jia-wei, et al. GPLAG: Detection of Software Plagiarism by Program Dependence Graph Analysis [C]// ACM SIGKDD'06. 2006: 872-881

[13] West A. Coping with plagiarism in Computer Science teaching laboratories[C]// Computers in Teaching Conference. Dublin, July 1995

[14] Gitchell D, Tran N. Sim: A Utility for Detecting Similarity in Computer Programs[C]// Proceedings 30th SIGCSE Technical Symposium. New Orleans, LA, USA, March 1999

[15] Ji Jeong-Hoon, Woo Gyun, Cho Hwan-Gue. A Source Code Linearization Technique for Detecting Plagiarized Programs[C]// ITiCSE'07. United Kingdom, June 2007

[16] Arwin C, Tahaghoghi S M M. Plagiarism Detection across Programming Languages[C]// Twenty-Ninth Australasian Computer Science Conference (ACSC2006). Australia, January 2006

[17] 赵长海，晏海华，金茂忠. 一个基于编译优化和反汇编的程序相似性检测方法[J]. 北京航空航天大学学报，2008，34(6)：711-715

[18] 边肇祺，张学工. 模式识别(第二版)[M]. 北京：清华大学出版社，2000

[19] Thonmas，等. 算法导论(第二版)[M]. 潘金贵，等译. 北京：机械工业出版社，2006

[20] Theodoridis S, Koutroumbas K. Pattern Recognition (Third Edition)[M]. Beijing: China Machine Press, 2006: 214-215

[21] 孟庆磊，姚春莲，宋建斌. 一种面向 H. 264/AVC 的快速帧内预测选择算法[J]. 北京航空航天大学学报，2007，33(2)：119-223

(上接第 151 页)

cardinality(1))…)

其中 Paper 中包含 1：*N* 的关联，该关联的类公理为：

Class ("WritePaper" partial restriction ("WritingPaper" allValuesFrom("Teacher") cardinality(1)) restriction ("WritenPaper") allValuesFrom("Paper") cardinality (1)))……

完成以上工作后，我们建立了模型的本体类公理模型，接下来将对该数据库中的数据记录进行抽取，并将其转换为本体实例。

对教师、课程实体分别抽取实例：

Individual(dboTeacher1 type(Teacher) value(StaffName "Barclay") value(StaffID 1)…)

Individual(dboCourse0 type(Course) value(CourseName "Philosophy") value(CourseID 0)…)

Teaching 表记录的实例为：

Individual (dboTeaching01 type (Teaching) value (Teaching-Course dboTeacher1) value(TaughtCourse dboCourse0) value (Credit "20") value(Period "0"))

对 Paper 表中的记录，按照 1：N 关联表记录的转换算法进行知识提取：

Individual (dboPaper0 type(Paper) value(PaperID "0") value (PaperName "ontologie review"))

Individual (dboWritePaper01 type(WritePaper) value(Writing-Paper dboTeacher1) value(WritenPaper dboPaper0))

结束语 关系数据的知识化是实现企业知识管理的重要部分。本文在形式化数据库记录模型和本体模型的基础上，研究了如何将关系数据转换为知识本体，提出了相应的信息提取算法，对其中的关键问题如标识和关联进行了重点研究，最后以实例对转换方法进行了展示。该算法生成的本体模型遵守 W3C 定义的 OWL DL 语法规则，因此具有良好的机器可读性和人机互操作性。本算法的提出可以使本体开发者更容易地将关系数据提取成为知识，为实现企业的知识管理提供了一种迅速构建本体的途径。同时，基于业务数据知识本体的创建也为企业下一步的知识进化及推理建立了基础。

参 考 文 献

[1] Lee Chang-shing, Kao Yuan-fang, Kuo Yau-hwang, et al. Automated ontology construction for unstructured text documents [J]. Data & Knowledge Engineering, 2007, 60(3): 547-566

[2] Neches R, Fikes R E, Gruber T R, et al. Enabling Technology for Knowledge Sharing[J]. AI Magazing, 1991, 12(3): 36-56

[3] McGuinness D L, van Harmelen F. OWL Web Ontology Language Overview. W3C Recommendation 20040210[OL]. http://www.w3.org/TR/owl-features/

[4] Dean M, Schreiber G. OWL Web ontology language reference. W3C Recommendation. 20040210[OL]. http://www.w3.org/TR /owl2ref/

[5] Patel-Schneider P F, Hayes P, Horrocks I, et al. OWL Web ontology language semantics and abstract syntax. W3C Recommendation. 2004202210[OL]. http://www.w3.org/TR/2004/REC-owl-semantics-20040210/

[6] Borst W N. Construction of Engineering Ontologies for Knowledge Sharing and Reuse[D]. University of Twente, Enschede, 1997

[7] Calvanese D, Lenzerini M, Nardi D. Unifying class-based representation formalisms[J]. Journal of Artificial Intelligence Research (JAIR), 1999, 11: 199-240

[8] 许卓明，董逸生，陆阳. 从 ER 模式到 OWL DL 本体的语义保持的翻译[J]. 计算机学报，2006，29(10)：1786-1795

[9] Zhang Guoqiang, Jia Suling. Ontology-based knowledge extraction for relational database schema[C]// 2009 Second International Symposium on Electronic Commerce and Security. 2009: 585-589

[10] 高复先. 信息资源规划——信息化建设基础工程[M]. 北京：清华大学出版社，2002

[11] Kingston J. Multi-perspective ontologies: Resolving common ontology development problems[J]. Expert Systems with Applications, 2008, 34(1): 541-550

[12] Chandra C A T. Organization and problem ontology for supply chain information support system [J]. Data & Knowledge Engineering, 2007, 61(2): 263-280

[13] Weng S-S, Tsai Hsine-Jen, Liu S-C, et al. Ontology construction for information classification[J]. Expert Systems with Applications, 2006, 31(1): 1-12