数据挖掘 实验报告*

刘扬[1+]

[1](南京大学 计算机科学与技术系,南京 210046)

Data Mining Final Assignment experiment export*

LIU Yang[1+]

[1](Department of Computer Science and Technology, NanjingUniversity, Nanjing 210023, China)

+ Corresponding author: Phn: +86-18832899861, E-mail: 541446436@qq.com

Abstract: Software clones has become a serious problem caused by developers reuse code by the copy-paste-modify operations or implement a functionality which are similar to an existing one. In the experiment, we are provided with many source code files written by the C language, and we need to judge whether two files are clone pairs. I did feature extraction on source code, and use the Random Forest, C4.5, KNN, SVM models on the training set and the testing set.

Key words: Radom Forest; C4.5; source code feature extraction; Integrated learning.

摘 要: 软件克隆现象的存在使得软件版权等受到侵害。此次实验在 c 语言源代码数据集进行克隆代码对的判断。我基于源码进行特征提取，并应用 weka 提供的 J48，Random Forest，Bagging 等方法进行克隆代码对的预测。最终，我发现集成版本的随机森林与源码特征提取结合会取得最高的分数。

关键词: 随机森林；C4.5；源码特征提取；集成学习

.

1 实验任务介绍

当开发者通过复制粘贴的操作复用代码, 或者实现与现有代码功能相似的代码时, 会构成软件克隆. 软件克隆会造成侵犯软件版权等行为, 因此, 以发现相似代码结构为目的的软件克隆检测应运而生. 设计一个可以自动检测两段代码是否构成克隆代码对的系统是很有必要的.

本次实验任务是一个真实的数据挖掘任务. 任务目标是预测两个源代码文件是否构成克隆代码对, 可以使用任意的数据挖掘方法, 目标是准确率越高越好.

在本次实验中, 待处理的代码数据构成两个不相交集合: 训练集和测试集. 训练集包括 83 个文件夹, 每个文件夹包含 500 个代码文件, 他们是用 c 语言实现的解决同一问题的代码, 也是被视为构成克隆代码对的文件. 不同文件夹内的代码文件互相不构成克隆代码对. 测试集包含 10000 个代码文件. 所有代码文件名都是独一无二的.

本次实验给出一个 csv 文件, 其中包含 20 万个代码对, 构成代码对的代码均来自于测试集. 任务目标是针对这 20 万个代码对, 判断它们是否是克隆代码对, 为克隆代码对添加标记 1, 非克隆代码对添加比较 0. 在 Kaggle 网站提交此 csv 文件, 将会得到评分. 评分使用 F1-score, 这意味着需要同时顾及查准率和查全率.

2 实验方法选择

本次实验可以分为提取特征向量和使用数据挖掘方法处理特征向量两部分. 对于数据挖掘方法, 我使用的是 weka 自带的分类方法, 通过测试集选择出效果比较好的算法. 实验难点在于特征向量的提取.

我构建了所有代码的抽象语法树, 但是由于部分代码并不是完整可执行的代码, 以及部分代码涉及C++的语法, 抽象语法树并不一定是正确的语法树.

观察到训练集具有一定的特殊性, 解决同一问题的代码都被视为克隆代码对. 从这一角度出发, 判断两段代码是否解决同一问题将对于判断克隆代码对具有很大的帮助.

我首先在抽象语法树中提取每个代码文件的常量字符串, 将所有的常量字符串总结在一个文件中, 并按照出现次数降序排列. 之后选择出现次数最高的若干字符串, 以此作为依据, 统计这些字符串在每个代码文件中出现的频率, 每个代码文件将得到一个稀疏向量. 将需要判断是否克隆的代码对应的向量相除, 一方为 0 则结果为 0, 若结果超过 1 则取倒数. 最终将相除的结果作为特征向量. 这种特征提取方法结合 weka 自带的分类方法，成绩大致在 0.25-0.35 之间。

除了提取常量字符串之外, 我还尝试了直接从源码构建特征向量. 对源码进行分词, 过滤掉某些不重要的词语，之后采取与提取常量字符串特征相同的方式处理. 最后的分数能达到 0.48。

对于数据挖掘挖掘方法, 我主要采用了 weka 自带的 J4.8, RandomForest, KNN 和 SMO, 在训练集上进行交叉验证, 效果最好的是 RandomForest 和 KNN, 在 kaggle 表现最好的是 RandomForest. 对于数据挖掘方法的选择， 根据我对于特征向量的提取方式，决策树这类的方法应该会有不错的效果。其他方法是我通过交叉验证根据正确率进行的选择。

除此以外, 我使用了两种基线算法, 分别是不使用数据挖掘方法进行分析和对代码中字符进行类似 onehot 编码再计算余弦距离, 效果都不尽人意.

3 方法实现细节

3.1 对常量进行特征提取

对代码文件建立抽象语法树，代码中的常量将在抽象语法树中用 constant 进行标注。将含有 constant 的一行从抽象语法树文件中读取出来，根据常量类型提取数字常量或字符串常量。由于数字类型的常量比较多，在提取常量时我只保留了字符串常量。字符串常量能有效的区分两段代码是否是克隆对。字符串常量主要包括两类，一类是程序中需要参与运算的字符串，另一类则是需要输出的字符串。例如输出 ABC3 个字母的全排列，参与运算的常量可能是 3 个单独的字母 A、B、C，输出的字符串可能是所有的排列情况，共有 6 个。有一些特殊的输出字符串如“YES”“NO”“Error”“Mon.”等将会非常有效的帮助我们判断两段代码之间的关系；除此以外，有一些 printf 中涉及格式化说明符的字符串可以告诉我们代码的输出格式，这对于判断两段代码关系也是很有效的。

我将这些重要的常量字符串进行统计，按照词频降序排列后保存在额外的文件中。之后在制作特征向量的时候，我选取总词频最高的若干项作为参考，计算每段代码的常量字符串在上述字符串中的出现次数，这样将会得到一个稀疏的向量，两个向量做差将会得到最终可用于训练的特征向量。特征向量的每一维代表着对应的高频字符串在两段代码中出现次数之比。

除了提取字符串之外，我还尝试了提取字符串和常量数字。常量对于判断克隆代码对很有帮助，但是从表现上来看，仅使用常量作为特征是不够的。

3.2 对源码进行特征提取

我使用空格符、括号等符号对原始代码进行分词，之后统计所有代码的分词结果，将所有代码中出现的词语按出现次数降序排列。从中选择出现次数最高的若干词语用于构造特征向量，例如，我们选取出现次数最高的 10 个词语，那么针对每个代码文件，将会得到一个 10 维向量，每一维的数字代表对应的词语在该代码文件中出现的次数；之后将需要对比的两端代码的向量相除，得到最终的特征

向量。在进行除法时，两个向量对应维度有一个是 0，则对应特征向量维度数字是 0；否则则是二者除法，选择介于 0 与 1 之间的结果。

这种特征提取方式与提取常量相同，但是除了常量之外，还提取到了操作符等内容，对于判断克隆代码对效果更好。

需要注意的是，这种方式同时也会提取到很多与判断克隆代码对无关的特征，需要我们对分词结果进行取舍。我的取舍策略是舍弃了所有包含“VAR”的变量名，之后选择若干维用于训练，得到了较好的效果。这里的取舍策略和选择多少维度都是没有严格指导的，需要经过多次尝试才可以达到最好的效果。

3.3 构建训练集的方法

如果我们穷举所有可能的代码对作为训练集，不仅会导致训练集过大，还会导致训练集严重不平衡，大多数样本的标记是 0。因此，使用随机抽样的方法构建训练集。具体而言，原始文件包含 83 个文件夹，每个文件夹包含 500 个文件，针对每一个文件夹，从中随机抽取 10 个文件，两两配对可以得到 45 个正样本，在每次配对的时候，随机选择其它一个文件夹，从中随机选择一个问价，配对构成负样本。这样最终得到 7000 多个样本，保证了正负样本个数为 1:1。

3.4 J4.8

J4.8 是一种决策树生成算法. 决策树是一种分类算法, 它采用树形结构, 使用层层推理实现最终分类, 是一种基于 if-then-else 的监督学习算法. 决策树是最简单的机器学习算法, 他易于实现, 可解释性强, 完全符合人类思维, 应用广泛.

决策树主要包含根结点、叶结点和内部节点。根结点代表训练集整体，叶结点对应决策结果，内部节点是通过以属性作为划分依据的划分后的训练集子集。决策树模型中最重要的部分是决策树的构造。 决策树采用自顶向下、递归生成的策略。如何对一个内部结点进行分裂，有着不同的选择标准，对应着不同的决策树生成算法。

最基础的决策树生成算法是 ID3，它使用信息增益作为属性选择的标准。通过引入信息论中熵的概念，可以计算每个结点的熵。通过计算分裂前后熵的差值得到信息增益。

假设当前训练数据集为 D，则熵的计算公式为：info(D) = $-\sum_{i=1}^{m} P_i \log_2 P_i$，其中 m 是输出类别个数，pi 代表输出为第 i 类的样本所占比例。通过属性 A 对 D 进行划分，将 D 划分成 v 个子集，则划分后的熵为 $\text{info}_A(D)=\sum_{j=1}^{v} \frac{|D_j|}{|D|} info(D_j)$；信息增益 gain(A)=info(D)- $\text{info}_A(D)$。ID3 在每次分裂时，贪心的选择具有最大信息增益的属性进行分裂。ID3 的缺陷主要是它倾向于选择属性有很多取值的属性进行分裂，因为这样的的属性会导致分裂尽可能得纯。

为了克服上述缺陷，C4.5(weka 中称为 J4.8)使用信息增益率作为分裂标准。首先定义分裂信息为 $\text{split\_info}_A(D)=\sum_{j=1}^{v} \frac{|D_j|}{|D|} info\left(\frac{|D_j|}{|D|}\right)$；进一步定义信息增益率为 gain_ratio(A)=$\frac{\text{gain}(A)}{split\_info A^{(D)}}$。除此之外，C4.5 还在构造过程中对树进行剪枝，还可以处理不完整数据和连续数据。

C4.5 的优点是分类规则易于理解，准确率高。缺点是它在构造过程中需要多次扫描和排序，导致算法比较低效，且算法只适用于能完全驻留内存的数据集。C4.5 比 ID3 更适合处理属性取值较多的数据。

3.5 Random Forest

随机森林是一种 Bagging 类型的集成算法。集成学习通过使用一系列的学习器进行学习，并将各个学习器通过特定的规则进行整合，从而获得比单个学习器更好的效果。

Bagging 通过在原始数据集上进行有放回的抽样产生多个新的数据集，使用分类器对新数据集分

别进行处理，最后通过投票或者求均值的方法统计所有的结果得到最终结果。Bagging 方法可以降低偏差和方差。bagging 方法通过随机抽样实现了样本随机性，并且在每轮随机采样中，训练集有大约 36.8% 的数据没有被采样，可以使用这部分数据检测泛化能力。

随机森林使用 cart 决策树作为基础的学习器，cart 是使用基尼指数系数作为选择标准的决策树生成算法。在随机森林中，每棵树选取的特征都仅仅是随机选出的少数特征，保证了特征随机性。随机森林具有较好的泛化能力和抵抗过拟合的能力。

随机森林的构造主要包括随机采样和完全分裂两部分。随机采样是指随机森林对输入数据的行、列分别进行有放回和不放回的抽样。完全分裂是指在形成决策树的过程中，决策树的每个节点要按照完全分裂的方式来分裂，直至结点不能分裂。

随机森林中每棵树的构造过程为：

1. N 表示样例个数，M 表示属性个数。
2. m 表示分裂时会使用的属性个数
3. 在样例中有放回的采样 N 次，构成新的训练集。
4. 对于每个节点，随机选择 m 个属性，设计最优划分方式。
5. 每棵树都不剪枝。
6. 随机森林中两棵树之间的相关性以及单独一棵树的分类能力是影响随机森林效果的两个重要因素。两棵树相关性越大，错误率越大。每棵树分类能力，错误率越低。

3.6 KNN

KNN 算法是一种用于分类和回归的非参数统计方法。算法的训练阶段只包括存储特征向量和标记。在分类时，K 是人工指定的超参数，一个没有类别标记的向量将被归类为最接近该点的 K 个样本点中最频繁使用的一类。

KNN 算法的核心思想是”近朱者赤，近墨者黑”。如果一个样本在特征空间中的 K 个最相邻样本中的大多数属于某个类别，那么该样本也属于这个类别。KNN 在确定分类时只依赖最邻近的 K 个样本，只与极少量样本有关。KNN 方法主要靠周围有限个临近样本、而不是判断类域的方法来确定所属类别，因此对类域的交叉或者重叠较多的待分类样本集来说，KNN 是比较合适的方法。

KNN 的执行过程如下:

1. 计算测试数据与各个训练数据之间的距离
2. 对距离从小到大进行排序
3. 选取距离最小的 k 个点
4. 确定前 k 个点类别的出现频率
5. 出现频率最高的类别作为预测分类

KNN 的优点是简单，易于理解和实现，无需估计参数和训练。KNN 适合对稀有事件分类，也适用于多分类问题。它支持增量学习，能对超多边形的复杂决策空间建模。KNN 缺点是计算量很大，分析速度慢。

3.7 SVM

SVM 的思想是通过一个线性超平面将样本空间划分成两部分实现分类的效果。SVM 希望样本点到超平面的最小距离尽可能得大，这样才能确保泛化能力比较好。样本距离超平面的最小距离被称为间隔。

SVM 的最大化间隔思想使用数学语言进行描述如下：

$$min_{\omega,b}\frac{1}{2}||\omega||^2, s.t.\, y_i(\omega^T x_i + b) \geq 1, i = 1, 2, \ldots, m$$

对于 SVM 的求解，定义拉格朗日函数

$$L(w,b,a)=\frac{1}{2}\|w\|^2+\sum_{i=1}^{m}a_i\left(1-y_i(w^Tx_i+b)\right)$$

使得 $a_i \geq 0$ ；$1-y_i(w^Tx_i+b)\leq 0$；$(1\leq i\leq m)$

SVM 满足强对偶性，可以通过解对偶问题得到 SVM 原问题的解。

SVM 的对偶问题是：

$$max_\alpha \sum_{i=1}^{m}\alpha_i-\frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m}\alpha_i\alpha_j y_i y_j x_i^T x_j$$

$$s.t.\sum_{i=1}^{m}\alpha_i y_i=0,\alpha_i\geq 0,i=1,2,\ldots,m$$

weka 中使用 SMO 算法高效求解 SVM 对偶问题。SMO 算法将上述问题分解成多个二次规划问题求解，每个子问题只需要求解两个参数，节省了时间成本和内存需求。

3.8 基线方法 1

方法 1 并没有使用到数据挖掘的方法，这也是我最早尝试的方法。我最初观察到代码中的常量字符串对于判断克隆代码对很有帮助，甚至可以说有决定性作用。例如，一段代码中提取到了“yes”，另一段代码中提取到了“Mon.”，我们几乎可以确信，这两段代码不是克隆代码对，这种字符串是包含语义信息的。除此之外，一段代码中出现了‘A’‘B’‘C’，另一段代码中出现了“ABC”，我们可以大概率确定它们是克隆代码对，但却没有很大把握，因为这样的字符串没有语义信息，不构成我们日常熟知的词语。还有一些代码，其中的字符串只包括格式化说明符，判断就更加困难，我们无法判断两段都包含“%d”的代码是否是克隆代码对。

根据上述观察，我通过对比两端代码中出现的常量字符串进行判断。如果两端代码中均出现了不是格式化说明符的字符串，则判定它们是克隆代码对。对于格式化说明符这一类的字符串，我分割所有的格式化说明符，使用余弦距离进行判断。

这种方法在训练集上分数可以达到 0.7 左右，但在测试集上测试效果十分不理想，只有 0.2 左右。

3.9 基线方法 2

受上一个基线算法启发，当两段代码拥有不同的常量时，常量中的字符往往也是不同的。方法 2 通过统计一段代码中所有常量（包括字符串和数字）的字符频率，由于字符个数是有限的，我们可以统计所有字符的次数，构建对应的向量。之后计算两个向量之间的余弦距离，通过不断调整判断的阈值，达到一个较好的分类效果。这种方法的效果比方法 1 还要差，在训练集上分数约为 0.5，在测试集上分数为 0.12 左右。

4 不同方法对比

	SVM	KNN	J4.8	随机森林	集成随机森林	基线算法 1	基线算法 2
常量特征提取	0.376	0.255	0.339	0.310	0.325	0.124	0.180
源码特征提取	0.292	0.342	0.349	0.451	0.483	0.177	0.217

4.1 与基线算法比较

从上表可以看到，不管使用哪种特征提取的方法，结合任意一种 weka 自带的数据挖掘方法进行预测，其效果都要比我自己实现的基线算法表现更佳。两种基线算法在训练集上表现不算太差，但是在测试集上效果很差，说明提取特征结合机器学习方法进行预测要更加适合处理本次实验的问题。

4.2 两种提取特征方法比较

通过纵向对比可以发现，对源码进行特征提取明显优于对常量进行特征提取。原因也很自然，对

于常量的特征提取只关注了代码中的常量，有些代码，其中的常量并不具有代表意义，这种情况下很容易出现误判；而对源码进行特征提取，考虑到了操作符、变量名等信息，其效果理应优于常量特征提取。

4.3 不同数据挖掘方法之间的比较

首先可以观察到，SVM 方法在常量提取上有着最高的分数，但这并不是正常的现象。特征向量每一维对应着一个高频词，其在不同代码中出现频率很可能不相同，因此使用决策树一类的算法将是较好的选择。至于为什么 SVM 在常量提取上有着异常高的分数，这和我一开始错误的构建训练集、测试集有关。最初的时候，我分别统计了测试集和训练集的高频词，分别构建对应的向量，这导致了测试集和训练集中相同向量的想同维度对应着不同的高频词。但是这样构造出来的向量可以看作不同代码与高频词数显次数之间的关系，因为高频词是按出现次数来排序的。这种情况并不适合决策树，但是却导致 SVM 性能优于其他的方法。在正确提取特征的情况下，SVM 在训练集和测试集上的效果都要低于其他数据挖掘分类方法，故 SVM 并不适合处理此问题。

通过横向对比，可以发现最优的方法是随机森林和它的集成版本。但是这两种方法在第一种热症提取方法上表现并不好，我认为这是第一种特征提取方式不够全面导致的。因此，对比不同的数据挖掘方法，我们选择源码特征提取下的分数作为比较依据。

对比发现，KNN 与 J4.8 表现接近，但是都明显落后于随机森林，集成的随机森林效果优于普通的随机森林方法。

5 对方法和结果的讨论

通过多次实验，可以发现，对源码进行特征提取，效果要优于对于常量进行特征提取。究其原因，常量特征提取不充分，并且我的处理方式也并非最好，导致其效果较差。对源码的特征提取，其描述能力一定大于常量特征，但其中有很多无用冗余的部分，包含这一部分进行数据挖掘，将会对判断克隆代码对产生干扰，因此，需要对源码分词后的结果进行筛选。总结起来，我的特征提取方法都仅仅局限在源代码层面，虽然我有构建抽象语法树，但是并未加以利用。从抽象语法树中，我们还可以进一步提取和结构有关的特征，由于结构特征提取较麻烦且可能包含部分领域知识，我并未进行这方面的探究。

不同的数据挖掘方法孰优孰劣，我觉得不仅仅和需要处理的问题有关，也和特征提取有着较大的关系，从 SVM 反常的表现就可以看出这一点。而具体哪一种数据挖掘方法更加适合此次问题，我是通过提取特征之后，在训练集上进行交叉验证得到的。

我最终分数是 0.48 左右，这个分数并不理想。通过网上查阅，我发现基于源代码进行特征提取，往届的学长可以达到 0.6 左右的分数。我觉得这个分数之间的差距来源于特征提取。特征提取是本次实验最难也是最重要的一步，在这一步我的处理并不够好，所以导致最后分数表现不佳。

References:

[1] 随机森林介绍 https://www.jianshu.com/p/a779f0686acc

[2] 随机森林特点 https://www.jianshu.com/p/ae1826eb7836

[3] 决策树介绍 https://easyai.tech/ai-definition/decision-tree/

[4] KNN 介绍 https://blog.csdn.net/lx85416281/article/details/40656877

[5] C4.5 的优点 https://blog.csdn.net/u010510549/article/details/41916441

[6] SVM 介绍 https://zhuanlan.zhihu.com/p/35755150