应用概率统计 第 35 卷
第 1 期 2019 年 2 月

Chinese Journal of Applied Probability and Statistics
Feb., 2019, Vol. 35, No. 1, pp. 86-108
doi: 10.3969/j.issn.1001-4268.2019.01.007

# 基于最大信息系数的软件缺陷预测模型 *

崔 军 刘亚娜
(山西大学计算机与信息技术学院, 太原, 030006)

郭新峰 王瑞波 李济洪*
(山西大学软件学院, 太原, 030002)

**摘 要:** 在软件缺陷预测的回归建模中, 由静态代码提取的类层面度量元 (特征) 以及由方法聚合 (sum、avg、max、min) 到类的特征往往较多, 使用传统的特征选择方法 (如 AIC、BIC) 通常先要确定了模型, 不同的模型选出的特征集差异较大, 且模型的可解释性差. 最大信息系数 MIC (maximal information coefficient) 是 Reshef 等 [4] 提出的度量两个连续变量之间相互依赖程度的一个指标, 且有基于观测数据的计算办法. 本文基于软件缺陷个数与各特征的 MIC 度量先选择特征, 再对所选特征进行了适当的幂次变换, 最后使用主成分泊松和负二项回归建模. 本文实验基于 NASA 的 KC1 的类层面数据集, 采用了 $m \times 2$ 交叉验证的序贯 $t$-检验来对两模型的性能差异的显著性进行检验, 模型性能评价指标采用 FPA、AAE、ARE. 实验结果表明: 1) 基于 MIC 选出的特征主要是 sum、avg、max 三种聚合模式特征, 与 AIC、BIC 方法有明显的差异; 2) 对特征做适当的幂次变换在多数模型下可以改善其性能; 3) 对特征做幂次变换后,做主成分分析与因子分析可以得到两个明显的因子, 其一个因子正好对应 avg 与 max 聚合模式的特征集, 另一个因子正好对应 sum 的聚合模式特征集, 使得模型具有较好的可解释性. 综合实验的各项指标可以得出, sum、avg、max 三种聚合模式对软件缺陷预测有显著作用, 且基于 MIC 所选特征而构造的模型是有优势的.

**关键词:** MIC 度量; 软件缺陷预测; 泊松回归模型; 负二项回归模型; $m \times 2$ 交叉验证序贯 $t$-检验
**中图分类号:** O212

**英文引用格式:** CUI J, LIU Y N, GUO X F, et al. Software defect prediction model based on maximal information coefficient [J]. Chinese J Appl Probab Statist, 2019, 35(1): 86–108. (in Chinese)

## §1. 引 言

计算机软件行业是一个飞速发展的产业, 长期以来人们过多地关注了这个产业带来的巨大的经济效益, 却忽略了计算机软件产业本身存在的问题. 直到大量的软件产品不能按时、按质、按量完成, 导致了 “软件危机” 的爆发, 软件质量问题才被越来越多的人关注. 如何提高软件质量, 降低软件中的缺陷成为近年来研究的主要问题之一. 提高软件质量的通常的方法就是软件测试. 测试工作贯穿于软件项目从设计、编码到维护的整个生命周期过程. 软件测试是软件开发的被动之举, 除此之外, 开发者希望在开发的早期就能够发现软件产品中的缺陷, 这称为软件缺陷预测. 在软件工程领域中软件缺陷是指可运行产品中会

*国家社科基金项目 (批准号: 16BTJ034) 资助.
*通讯作者, E-mail: lijh@sxu.edu.cn.
本文 2018 年 4 月 12 日收到, 2018 年 10 月 25 日收到修改稿.

导致软件失效的瑕疵[1]. 开展软件缺陷预测工作的根本目的是为了提高软件产品的可靠性和有效性, 提高发现和排除软件缺陷的效率.

软件缺陷预测的较早研究可以追溯到 Briand (1992). 在通常的软件产品开发过程中, 软件开发组织只能通过软件测试结果来了解软件产品缺陷的状况. 但软件测试成本高昂, 且只有在开发过程中和开发工作完成后才可进行, 那时开发组织往往已没有足够时间排除缺陷. 另外如果软件开发组织经过测试发现软件产品的缺陷数量或缺陷率高于所要求的指标, 那么它们只能推迟产品发布时间来弥补现有缺陷, 或按时发布包含着大量缺陷的软件产品. 为此研究人员提出了许多软件缺陷预测方法.

软件缺陷预测理论假设在高复杂度模块具有高缺陷率的前提下, 用软件产品的一些度量 (metrics) 来表征软件产品的复杂程度, 进而对软件模块的缺陷状况进行预测. 根据预测结果, 软件开发组织可以将有限的开发、测试资源集中于容易出现缺陷的高风险模块, 从而更有效地发现和排除缺陷, 提高软件产品的质量和可靠性. 研究表明, 软件缺陷预测能够帮助提高软件的可靠性和质量[2]. 软件缺陷预测基于软件产品的多种度量. 在软件产品度量中, 度量可以分为静态度量和动态度量两类[3]. 静态度量可以通过对软件产品的静态统计得到, 如代码行数、调用过程的个数等等; 动态度量则是指在产品运行过程中采集的非静态度量, 其需要在软件产品开发过程中和开发工作完成后经测试才能获得. 基于静态度量的缺陷预测的目的是帮助软件开发者在较早期发现软件产品中的缺陷, 因此, 更加受到软件开发者的欢迎. 本文将这些度量统称为特征.

近年来统计学习理论的发展对软件缺陷预测研究领域提供了新的思路和方法, 越来越受到许多学者的广泛关注. 泊松和负二项回归模型是目前较好的软件缺陷预测模型, 但其性能显著依赖于特征选择. 一般来说, 基于静态代码可抽取的特征较多, 通常的特征选择方法选出的特征集往往可解释性差. 直观上可以从所有特征出发, 使用主成分降维, 但由于特征之间相互关系较为复杂, 而主成分只能压缩其间的线性关系, 往往难以有效降维. Reshef 等[4] 提出了两个变量之间相依程度的 MIC 度量. 不同于相关系数只能度量两变量的线性相关程度, MIC 不仅能够度量变量间的线性相关关系, 还可以度量变量之间非线性函数关系, 且 MIC 值不受变量做任意函数变换的影响. 也即对给定的观测数据, 两个变量的 MIC 在观测数据做任何函数变换下, MIC 值均不变. 本文正是想利用 MIC 这个性质, 在对软件缺陷数建模中, 先选择与软件缺陷数有较大的 MIC 的特征 (度量元), 这个特征选择的步骤与使用什么模型无关, 不论后面建模中使用什么模型以及如何使用这些特征, 理论上均不会丢失真正有相依关系的特征. 其次对所选的初始特征集中的特征进行了适当的幂次变换, 使得变换后的特征与缺陷数指标尽量接近线性, 然后再对变换后的特征集进行主成分分析 (principal component analysis, 简称 PCA), 最后基于得到的主成分做泊松和负二项回归模型, 并通过实验与传统的特征选择方法, 如 AIC、BIC, 进行了性能的对比.

本文结构如下, 第一、二节主要介绍了软件缺陷度量方面的背景知识及研究现状; 第三节主要描述了基于 MIC 度量的特征选择; 第四节主要介绍了软件缺陷预测的负二项回

归模型; 第五节介绍了模型性能比较的 $m \times 2$ 交叉验证序贯 $t$-检验; 第六节介绍了实验的过程; 第七节是本文结论.

## §2. 软件缺陷预测的研究现状

近年来, 有许多基于机器学习模型的软件缺陷预测研究. 目前常用的软件缺陷预测模型主要有分类和回归两类, 主要有 Logistic 回归 (LR)、分类回归树 (CART)、多元线性回归 (MLR)、人工神经网络 (ANN)、随机森林 (RF)、泊松回归 (PRM)、负二项回归 (NBRM) 等. 使用的数据绝大部分来自 NASA 和 PROMISE 数据库, 相关文献也很丰富, 比如, 文献 [5–13] 等. 总结文献的结果, 概括地说, 预测性能较好的模型是随机森林 (见文献 [6]), 以及文献 [7] 的多个机器学习模型的集成方法, 但显然这些模型可解释性较差.

在特征选择方面, 根据 Khoshgoftaar 等[8] 在 16 个数据集上的比较结果, 较好的特征选择方法是信息增益. Okutan 和 Yildiz[9] 在 9 个 PROMISE 数据库上的数据集上进行实验并且发现最有效的特征为 Lines of Code、Response for Class 和Lack of Coding Quality. 但使用不同的模型、不同的数据集选到的特征差异较大, 特征选择结果依赖于模型, 若选用了不合适的模型, 就可能丢掉有用的特征.

目前的文献中大部分是基于分类数据, 而基于回归数据建模由于可用的数据集有限, 文献中相关研究相对较少. 另外, 由于分类数据中有缺陷的模块少, 虽然可以使用不均衡分类算法, 但效果也不太理想, 这点也驱使研究人员希望使用回归来建模. 通常来说, 回归数据是分类数据的聚合, 其中的响应变量使用的是模块 (或类) 含有缺陷的个数, 而不是本模块是否有缺陷 (0.1 值), 特征的取值也是聚合之后的, 理应包含更多的建模信息, 应当得到更好的预测模型 (至少在总量预测上). Koru 和 Liu[14] 使用了 KC1 类层面的数据, 把特征从方法级聚合[15] (sum、avg、max、min) 到更大的粒度 (类) 上来建模, 通过实验说明由方法级聚合到类级别后, 模型总的预测性能有所提高, 但并没有指出什么样的聚合特征可以显著提高模型性能. 而 Zhang 等[15] 采用了 11 种聚合方式, 在四种不同类型的软件缺陷预测模型上进行实验, 最终结果显示不同类型的软件缺陷模型性能最好时所采用的聚合方式是不同的.

Ostrand 等[16] 使用负二项回归建立软件缺陷回归模型. Gao 和 Khoshgoftaar[11] 也使用了包括泊松回归、负二项回归等多个模型, 并使用数据重抽样的交叉验证 (RLT)、似然比检验和 Tukey 的多重比较技术进行了多方面的分析. 就目前文献结果看, 负二项回归是回归建模中最好的模型. 事实上, 负二项回归模型可以看做是对泊松回归的拓展, 其方差不再等于期望, 而是期望的单增函数, 且含有新的待估参数. 理论上说, 负二项模型的参数估计完全可以通过极大似然来估计, 但由于似然函数复杂 (有伽马分布的密度函数), 求导难. 实用中, 需要先估计新的待估参数以及设定合适的控制参数, 否则常常会导致极大似然估计不收敛, 使得模型性能下降. 泊松回归模型是负二项回归的特殊情形, 但其参数估

计要简单的多. 实用中, 一般先对数据拟合泊松回归, 检验方差是否大于期望, 即是否存在 overdispersion, 若存在, 再进一步使用负二项回归建模.

概括地说, 就软件缺陷预测回归建模中, 存在的主要问题是: 1) 传统的特征选择方法, 如 AIC[17]、BIC[18], 依赖于使用的模型. 不同的模型、不同的数据集选到的特征差异较大, 最终得到的模型的可解释性较差. 若采用的模型不适当就可能丢掉有用特征. 能否找到一种与模型无关的有效的特征选择方法; 2) 回归建模中主要使用的是聚合特征 (使用如 max、avg、sum、min 聚合函数获取的新特征), 那么哪种聚合对回归建模是有显著作用的?

为此, 本文以静态特征的类层面特征以及聚合特征为基础, 先采用 MIC 度量预选特征, 再对特征做适当的幂次变换, 之后使用主成分降维, 最后使用主成分泊松和负二项回归建模, 进一步做因子分析, 使得模型有较好的可解释性. 模型对照的统计检验采用了最新的 $m \times 2$ 交叉验证的序贯 $t$-检验, 并在 NASA 的 KC1 的类层面聚合数据集上进行了实验.

## §3. 基于 MIC 的特征选择

### 3.1 两个随机变量的相依性 MIC 度量

关于两个随机变量的相依性度量, 常见的主要有相关系数、Spearman 秩相关 (SRCC)、互信息等. 其中相关系数、SRCC 主要是度量两个随机变量之间的线性相关程度, 对一般的非线性相依关系并不有效. 互信息是度量任意两个随机变量相依性的较好度量, 但在具体计算时, 只能对离散随机变量, 对连续变量的一组观测, 还没有具体的计算方法.

Reshef 等[4] 给出两个随机变量的 MIC (maximal information coefficient) 度量, 拓展了互信息的概念, 并对连续变量的观测数据情形下给出了有效的算法. MIC 可以度量多种类型的相依关系, MIC 是基于两个随机变量 $n$ 个观测值来计算的, 其定义简单描述如下:

给定两个随机变量的观测值 $D:(X,Y)$, 把 $x-y$ 平面划分成若干个小网格使得数据集 $D$ 中的所有的点都落入网格 $G$ 中, 这样就把 $D$ 化为离散的变量可以计算其互信息. 不同的网格大小计算的互信息不同, 取所有可能网格中最大的互信息, 再做归一化就得到 MIC.

**定义 1**　给定两变量 $X,Y$ 的 $n$ 个观测值 $D$, 那么这个随机变量的 MIC 定义为

$$\mathrm{MIC}(X,Y\mid D)=\max_{(x,y):xy<B}\frac{\max\limits_{G\in\Omega(x,y)} I(X(G),Y(G))}{\ln\min(x,y)}. \tag{1}$$

这里, $\Omega(x,y)$ 表示大小为 $x\times y$ 的二维网格集合, $X(G),Y(G)$ 表示基于网格 $\Omega(x,y)$ 的离散变量, $I(X(G),Y(G))$ 表示其互信息, $B$ 为最大网格数, Reshef 等[4] 建议取 $B=n^{0.6}$.

MIC有如下性质:

显然, $0\leqslant \mathrm{MIC}\leqslant 1$, 如果 $X$ 与 $Y$ 相互独立时, 那么 $\mathrm{MIC}(X,Y)=0$; MIC 有对称性, 即 $\mathrm{MIC}(X,Y)=\mathrm{MIC}(Y,X)$;

当两个变量 $Y, X$ 存在 $Y = f(X)$ 时, $\mathrm{MIC}(X, Y) = 1$, 也就是说, 对于两个有确定的函数关系随机变量, 不论这个函数关系是什么样的, MIC 的值都是 1;

若对变量 $X$ 做任何单调函数 $g(X)$ 变换, MIC 不变, 即 $\mathrm{MIC}(X, Y) = \mathrm{MIC}(g(X), Y)$;

从 MIC 的定义可以看出, 与相关系数相比 MIC 稳健性好, 即 MIC 不受异常值的影响, 而相关系数却易受异常值的影响.

Reshef 等[4] 还给出了 MIC 度量的显著检验, 也即, $\mathrm{MIC}(Y, X)$ 达到多大可以认为两个变量 $Y, X$之间存在显著的相依关系, 并构造了统计检验的 MIC 临界值表.

### 3.2 基于 MIC 的特征选择

从上述分析知道, MIC 度量了两个变量之间的更为一般的相依关系. 因此, 类似于采用互信息度量来选择变量, 同样可以采用 MIC 度量来选择变量.

Xu 等[19] 采用 MIC 进行了选择变量, 方法是先计算特征和分类响应变量 $Y(0, 1)$ 间的 MIC, 然后依照降序排列选出前 $p$ ($p = \lfloor m/\ln m \rfloor$, $m$ 为所有特征) 个特征. 这种方法的缺点在于 $p$ 的确定缺乏依据, 从其选择的形式 $p = \lfloor m/\ln m \rfloor$ 上看, $p$ 的确定似乎是借鉴高维稀疏特征情形下的有效特征数来确定的, 但我们认为软件缺陷预测的特征还称不上高维稀疏特征, 应该使用 MIC 的显著性判别的临界值表来确定 $p$; 另外, Xu 等人的方法是针对二分类响应变量, 但我们认为 MIC 更适用于度量连续变量的相依性. 因此, 本文使用 MIC 进行特征选择的具体步骤如下:

1) 逐个计算每个特征 $x_i$ 和响应变量 $Y$ (这里就是 NUMDEFECTS) 之间的 MIC 值;

2) 基于 MIC 度量的显著性检验的临界值表, 选出 $\mathrm{MIC}(x_i, Y)$ 大于临界值的所有特征.

### 3.3 MIC 特征选择方法更适用于软件缺陷预测数据

1) 传统的特征选择方法, 如 AIC、BIC, 是以似然函数加惩罚项为目标函数, 求解最小的特征集合. 在特征较多情况下, 一般使用 Forward (or Backward) Stepwise 的贪心算法, 逐步往模型中增加 (或减少) 特征, 每步增选的特征是挑选目标函数增加最快的特征, 但在许多模型下, 其本质上是挑选以当前拟合模型的残差与候选特征中的最大线性相关的特征. 因此, 若候选特征与缺陷数有稍微复杂的非线性依赖关系, 很有可能被排除在模型外.

2) 一般的相关系数对变换很敏感. 例如, NASA 的 KC1 回归数据集中的缺陷数 NUMDEFECTS 与聚合特征 maxNUM_OPERATORS 之间, 从图 1 中很难看出两者的相依性. 用相关系数度量为 0.380764, 对两个变量都做 $\ln(x + 1)$ 变换, 得到相关系数为 0.538815, 说明相关程度对采用的变换很敏感, 然而, 建模时我们并不知道该做什么变换. 而无论做什么单调变换, 两者的 MIC = 0.47221, 在 0.01 水平下是显著的, 却能更好地度量两者的相依关系. 因此, 用 MIC 来选特征更适用于复杂的软件缺陷的数据, MIC 度量对做任何单调函数变换均不受影响.

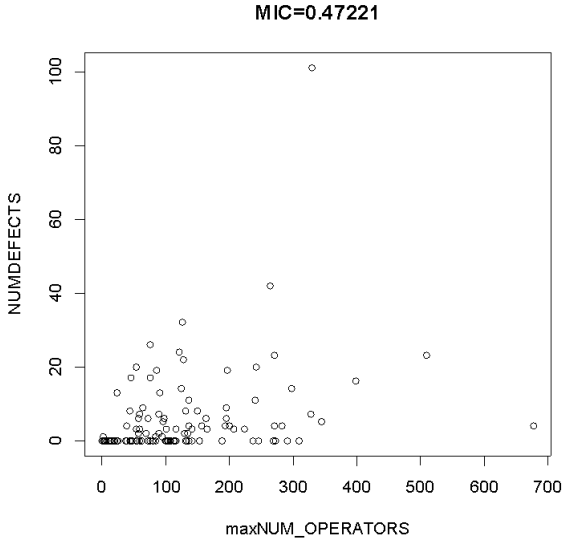


**图 1　maxNUM_OPERATORS 与 NUMDEFECTS 的散点图**

### 3.4　特征的幂次变换

为什么要对特征进行变换? 主要从如下两点考虑:

1) 从缺陷数与每个特征 $x_i$ 的散点图看出, 当缺陷数以及特征 $x_i$ 的值较大时比较分散, 统计上可以认为是 $x_i$ 的方差不稳定, 因此, 做方差稳定的幂次变换, 对广义线性模型 (负二项回归、泊松回归) 的拟合是有利的.

2) 利于使用主成分降维. MIC 特征选择方法由于只考虑每个特征 $x_i$ 与 $Y$ 的相依程度作为指标, 没有考虑特征之间的相互关系, 因此, 通常基于 MIC 的特征选择得到的集合包含较多的特征. 如何去除冗余特征, 或减小特征之间的相依关系, 使得软件缺陷预测模型既简单又有好的可解释性. 显然, 首先能够想到的是对所选特征集使用主成分降维, 但由于主成分只对特征之间的线性相关可以有效降维, 而对非线性相依关系效果不佳.

为此, 我们提出先对所选特征使用幂次变换, 也即对所选特征 $x_i$ 做变换:

$$f(x_i)=\begin{cases}\dfrac{x_i^{\alpha}-1}{\alpha}, & \alpha\neq 0,\ \alpha\in(0,2];\\ \ln(x_i), & \alpha=0.\end{cases}\tag{2}$$

选择 $\alpha$ 得: $\alpha_i^*=\arg\max\rho(f(x_i),\ln Y)$.

这里 $\rho(f(x_i),\ln Y)$ 为变换后的特征 $f(x_i)$ 与 $\ln Y$ 的相关系数, 也就是希望经过幂次变换的 $x_i^{\alpha}$ 与 $\ln Y$ 最大可能接近线性相关, 从而使得 $x_i^{\alpha}$ 之间的线性相关性增强, 这样基于 $x_i^{\alpha}$ 再做主成分降维才能更加有效. 这里使用 $x_i^{\alpha}$ 与 $\ln Y$ (而不是 $Y$) 的相关系数主要是考虑泊松和负二项模型本质上是对 $Y$ 取对数后与特征建立模型.

实用中, 我们将 $\alpha \in (0,2]$ 的取值区间按 0.05 间隔分为 40 个值 (0.05, 0.1, 0.15, ..., 1.95, 2), 还有对数变换 $\ln x_i$, 共 41 个. 从这 41 个值中选取使得 $x_i^\alpha$ 与 $\ln Y$ 之间相关系数最大来确定 $\alpha$.

## §4. 软件缺陷预测的回归模型

软件缺陷预测的回归模型是要预测一个程序文件中的缺陷个数. 通常一个程序文件是由多个小的程序文件组成, 大程序文件的缺陷个数是由每个小程序文件的缺陷聚合而来. 比如, 在面向对象的程序设计中, 类程序文件是由类文件中的各个方法文件汇聚而成, 若每个方法文件有类别标签 (有缺陷或没有缺陷), 且每个方法文件有没有缺陷是服从伯努利分布的随机变量, 若类文件中方法文件的个数 (聚合的个数) 是服从泊松分布的随机变量, 则聚合后的类文件的软件缺陷个数也服从泊松分布. 也即, 假定 $S \sim \text{Poisson}(\lambda)$, $Y_i \sim \text{Bernoulli}(p)$, $i = 1, 2, \ldots, S$, i.i.d., 则随机变量

$$Y \triangleq \sum_{i=1}^{S} Y_i \sim \text{Poisson}(\lambda p),$$

且 $\mathsf{Var}(Y) \leqslant \mathsf{Var}(S)$.

记 $Y$ 为软件缺陷数 (响应变量), $x = (x_1, x_2, \ldots, x_p)^\mathsf{T}$ 为自变量. 其泊松回归模型为

$$\begin{cases} Y \,|\, x \sim \text{Poisson}(\mu), \quad \mathsf{E}(Y \,|\, x) = \mu, \\ \text{LINK} : \ln \mu = x^\mathsf{T}\beta. \end{cases} \tag{3}$$

负二项回归模型是从泊松回归演变出来的, 在负二项回归模型中, 是假定 $Y \,|\, x$ 的条件期望等于 $\mu\epsilon$, 而 $\epsilon$ 是个随机变量, 且服从均值为 1 的伽马分布, 则 $Y \,|\, x$ 服从负二项 (negative binomial, 简写为 NB), 即负二项分布是由泊松分布与伽马分布的混合而得. 也即, 假定 $Y \,|\, x$, $\epsilon\text{Poisson}(\mu\epsilon)$, $\epsilon \sim \text{Gamma}(\theta, \theta)$, 这里 $\theta$ 为分布的形状参数和刻度参数, 则 $Y \,|\, x \sim \text{NB}(r, p)$. 若记 $\mu \triangleq r(1-p)/p$, $\theta \triangleq 1/r$, 则 $\mathsf{E}(Y \,|\, x) = \mu$, $\mathsf{Var}(Y \,|\, x) = \mu + \theta\mu^2$.

负二项回归模型为

$$\begin{cases} Y \,|\, x \sim \text{NB}(r, p), \quad \mathsf{E}(Y \,|\, x) = \mu, \quad \mathsf{Var}(Y \,|\, x) = \mu + \theta\mu^2, \\ \text{LINK} : \ln(\mu) = x^\mathsf{T}\beta. \end{cases} \tag{4}$$

这里的 $\theta$ 也称为模型的散度参数. 当 $\theta$ 趋近于 0 时, 就可以趋近于泊松回归模型, 因此, 泊松回归是负二项回归的特殊情形.

负二项回归模型有很多变种, 如 $\mathsf{Var}(Y \,|\, x) = \mu + \theta\mu$ 的 NB1, 这里的模型就是多数文献中的 NB2 模型.

### 4.1　模型参数的估计

负二项 (NB2) 回归模型的参数估计一般采用极大似然估计, 其对数似然函数为

$$L(\beta;\theta)=\sum_{i=1}^{n}\Big\{y_i\ln\Big[\frac{\theta\exp(x_i^{\mathrm{T}}\beta)}{1+\theta\exp(x_i^{\mathrm{T}}\beta)}\Big]-\frac{1}{\theta}\ln[1+\theta\exp(x_i^{\mathrm{T}}\beta)]\\+\ln\Gamma\Big(y_i+\frac{1}{\theta}\Big)-\ln\Gamma(y_i+1)-\ln\Gamma\Big(\frac{1}{\theta}\Big)\Big\}.$$

本文采用的是 R 软件的 mass 包中的 glm.nb(), 使用时需要设定好控制参数, 否则常常不能收敛. 也可使用 $\mathrm{glm}(\cdot,\mathrm{family}=\mathrm{negative.binomial}(\theta))$, 此时 $\theta$ 需要事先设定. 若设置 family = Poisson 就是泊松回归.

### 4.2　散度参数的假设检验

对散度参数 $\theta$ 可以进行假设检验 $H_0:\theta=0$ vs $H_1:\theta>0$, 若不能拒绝原假设, 则直接使用泊松回归建模. 下面这个检验来自文献 [20].

似然比检验: $G^2=-2(L_{\mathrm{PO}}-L_{\mathrm{NB2}})\sim\chi^2(1)$. 检验采用单边检验. 当 $G^2>\chi^2_{1-2\alpha}(1)$ 时, 则拒绝原假设而接受备择假设, 认为散度参数 $\theta$ 大于零, 预测模型应当采用负二项回归. 这里 $L_{\mathrm{PO}}$ 表示泊松回归的对数似然函数, $L_{\mathrm{NB2}}$ 表示负二项回归的对数似然函数, $\chi^2_{1-2\alpha}(1)$ 表示 $\chi^2(1)$ 分布的 $1-2\alpha$ 分位数. 注意, 这个检验应当在训练集上进行.

### 4.3　主成分

主成分 (principal component analysis, 简称 PCA) 方法能够有效地去除特征之间的相关性, 提高缺陷预测模型的性能.

记 $X_{n\times p}$ 为中心标准化的设计矩阵, 也即, $X_{n\times p}$ 中的列由变换后的特征 $f(x_i)$, $i=1,2,\ldots,p$ 构成, 且经过了中心标准化, 这里 $p$ 为所选特征的个数. 记 $Z_{n\times k}=X_{n\times p}U_{p\times k}$ 为前 $k$ 个主成分 ($k<p$), $U_{n\times k}\Sigma_{k\times k}$ 称为载荷矩阵, $\Sigma_{k\times k}$ 为对角阵, 其元素为特征根的开方.

传统的主成分个数 $k$ 的选取是采用累积贡献率来确定 $k$, 但贡献率仅仅是从主成分表示特征矩阵全部列的 “信息” 的比例来确定, 不是从模型的预测性能角度. 但本文最终目的是建立软件缺陷的预测模型, 应当采用缺陷预测的性能指标 (本文用 FPA) 在测试集上最大来选.

本文采用一组 $3\times2$ 交叉验证[21] 来选择 $k$. 另外, $k$ 的选择不能与模型参数估计在同一组交叉验证上, 因为要先确定 $k$, 才能去估计模型参数. $k$ 的确定与模型的参数估计有先后顺序, 在同一组交叉验证上容易导致对同一组数据的过适应 (adaptive). 为此, 本文采取了两组 $3\times2$ 交叉验证方案, 第一组用于 $k$ 的选择, 第二组用于模型参数估计以及模型性能对照的 $t$-检验. 这样做减少了对数据的 “adaptive”, 使得结果可信度增强.

## §5. 两模型性能对照的 $m \times 2$ 交叉验证序贯 $t$-检验

在本文 4.2 中有模型的拟合度的检验, 是在数据的训练集上拟合模型时判断哪个模型拟合的更好, 但拟合的好不能保证预测的好. 软件缺陷预测更注重预测的效果, 需要在测试集上判断模型预测的好坏. 由于软件缺陷预测所用的数据集一般比较小, 因此, 模型预测性能常常使用交叉验证的方法. 而采用交叉验证时, 最重要的是给出模型性能指标的合理的方差估计.

数据总是带有误差的, 采用合理的统计显著性检验是得到可靠结论的保障. 在软件缺陷预测的大部分文献中, 采用的多是基于 5 折 (10 折) 的交叉验证的统计检验. 但正如 Wang 等 [21] 指出, 通常的基于 5 折 (10 折) 交叉验证的统计检验是激进的, 因为, 其检验的方差估计是在假定 5 次训练、测试是相互独立, 采用 5 次试验的样本标准差得到的, 这往往造成其真实没有显著差异而推断有显著差异. 为此, Wang 等 [22] 分析了常用的模型性能对比的 $t$-检验存在的问题, 提出模型性能比较的正则化 $m \times 2$ 交叉验证序贯 $t$-检验方法. 理论和模拟验证该检验是相对比较保守的检验, 可以得到更为可信的结论. 下面以 $m = 3$ 为例来做简要介绍, $3 \times 2$ 交叉验证就是随机将数据集 $D$ 划分成大小相同的 4 份, 然后任取两份作为训练集, 其它两份作为验证集, 进行 2 折交叉验证, 这样不同的组合共有 3 组, 总共实施 3 组 2 折交叉验证 (而不是做 4 折交叉验证), 可以得到 6 个结果. Wang 等 [22] 给出了一般情形下的 $m \times 2$ 交叉验证的正则化数据切分算法, 以及性能指标的方差估计. 显然, Wang 等 [21]所给的 $3 \times 2$ 交叉验证 $t$-检验自然可以扩展为相应的两模型性能比较的 $m \times 2$ 交叉验证序贯 $t$-检验, 简单描述如下:

给定两个机器学习模型 A、B 及模型评价指标 $\mu_A$、$\mu_B$, 记 $\mu = \mu_A - \mu_B$ 为两个模型性能之差, 不妨假设 $\mu_B$ 为基线 (baseline) 模型的性能.

原假设: $H_0 : \mu \leqslant 0$; 备择假设: $H_1 : \mu > 0$.

记 $\mu$ 的在数据集 $D$ 上的第 $i$ 次切分上的 2 折交叉验证估计为

$$\widehat{\mu}_i = (\widehat{\mu}_1^{(i)} + \widehat{\mu}_2^{(i)})/2, \qquad i = 1, 2, \ldots, m,$$

$\mu$ 的 $m \times 2$ 交叉验证估计为 $m$ 次 2 折交叉验证估计的平均, 记为 $\widehat{\mu} = \sum\limits_{i=1}^{m} \widehat{\mu}_i/m$, 其检验统计量如下式所示 [23]:

$$T_{m\times 2} = \frac{\widehat{\mu}}{\widehat{\sigma}} \sim C_m \cdot t(2m-1). \tag{5}$$

这里, $\widehat{\sigma}^2 = (2m)^{-1} \sum\limits_{i=1}^{m} \sum\limits_{k=1}^{2} (\widehat{\mu}_k^{(i)} - \widehat{\mu})^2$ 为 $\widehat{\mu}$ 的方差的保守估计 [22], $C_m = \sqrt{(2m+1)/(2m-1)}$ 使得序贯 $t$-检验的判决更为可靠 [23].

文献 [23] 证明了 $T_{m\times 2}$ 近似服从自由度为 $2m-1$ 的 $t$-分布. 当 $\widehat{\mu} > C_m \cdot \widehat{\sigma} \cdot t_\alpha(2m-1)$ 时, 就拒绝原假设, 接受备择假设. 这里 $t_\alpha(2m-1)$ 为 $t$-分布的 $\alpha$ 上侧分位数. 通常我们推

荐最小的 $m$ 取 3, 之后, 再逐步增加实验次数 $m$. 若到一个较大的实验次数 (比如 $m = 12$), 仍然没有停止, 可强行停止. 关于实验停止次数 $m$ 的选取, 可参看文献 [23] 中的论述.

# §6. 实　　验

实验目的:

1) 比较 MIC、AIC 和 BIC 特征选择的差异性;

2) 对特征做幂次变换的必要性分析;

3) 特征的主成分分析之后的因子分析, 因子的可解释性分析;

4) 哪种类型的聚合模式可以显著提升模型性能?

实验数据: 在 NASA 的 KC1 类层面数据集 (每行是一个类的特征数据) 上.

## 6.1 软件缺陷预测的性能度量指标

1) 平均绝对误差 (AAE)

软件缺陷预测领域, 将平均绝对误差 (average absolute error, AAE) 作为对软件预测方法进行评价的一个指标, AAE 有如下定义:

**定义 2**

$$\mathrm{AAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \widehat{y}_i|, \tag{6}$$

其中 $\widehat{y}_i$ 表示第 $i$ 个模块的预测值, $y_i$ 表示相应的实际值, $n$ 表示模块的数量.

从公式中可以看出, 如果每个模块的预测值和实际值相差越小, 则得到的 AAE 越小, 也就说明这种预测方法和实际情况相差较小, 预测的精度较高, 也就说明了这种预测方法的效果好.

2) 平均相对误差 (ARE)

平均相对误差 (average relative error, ARE) 也是一个常用来评价回归模型预测效果的指标. 所谓相对误差, 就是真实值和测量值之间的差距和真实值的比值, 平均相对误差, 就是把 $n$ 次测量得到的相对误差加起来之后再求平均值. 在评价软件缺陷预测方法的效果时, 也会使用到平均相对误差, 并将其定义为:

**定义 3**

$$\mathrm{ARE} = \frac{1}{n}\sum_{i=1}^{n}\left|\frac{y_i - \widehat{y}_i}{y_i + 1}\right|. \tag{7}$$

公式中也可以看出, 如果测量值与真实值之间的差距越小, 真实值的数值越大, 得到的相对误差就越小. 相对误差考虑了实际值的大小对预测精度的影响, 因此, 它比平均绝对误差方法要符合实际.

3) FPA

FPA 实际上是前 $m$ 个模块 (预测出来的缺陷数按从大到小的顺序排列得到的前 $m$ 个) 预测出来的缺陷数占实际总缺陷数比例的均值, 前 $m$ 个模块预测的缺陷占实际缺陷的比例将其定义为

$$\frac{1}{n}\sum_{i=k-m+1}^{k} n_i, \tag{8}$$

$n_i$ 是指模块 $i$ 的实际缺陷数, $n$ 为 $k$ 个模块总的缺陷数目, $k$ 为模块数.

FPA 定义为:

**定义 4**

$$\mathrm{FPA} = \frac{1}{k}\sum_{m=1}^{k}\frac{1}{n}\sum_{i=k-m+1}^{k} n_i. \tag{9}$$

FPA 是一个从模型预测缺陷数的顺序考虑的性能度量指标, 只要预测的顺序与实际缺陷数的顺序一致, 模型的预测性能就好. FPA 越高该预测模型性能越好.

## 6.2 实验数据

本文实验数据来源于 PROMISE 平台上的 KC1 数据集, 可从网址 http://openscience.us/repo/defect/ 中下载.

KC1 是 NASA 的某宇宙飞船控制系统的地面数据子系统, 该子系统使用 C++ 语言实现. 2005 年 5 月, PROMISE 公布了子系统的缺陷数据集, 被广泛用于软件缺陷预测研究领域. KC1 数据集包含 2 109 个模块的缺陷趋势 (1, 0 值表示缺陷的有无), 每个模块包含 21 个静态代码特征, 源自 21 个度量指标, 包括: 5 个不同的代码行数 (LOC) 指标, 如总代码行数、可执行代码行数、仅注释行数、空行数、含注释代码行数; 3 个 McCabe 复杂度度量指标; 12 个 Halstead 度量指标; 1 个分枝数 (branch-count) 指标. 基于上述 21 个传统的静态代码特征建立的数据集, 被广泛用于软件缺陷预测模型的建模研究.

PROMISE 公布的第二个 KC1 数据集包含 145 个类的缺陷个数, 每个类包含 94 个代码特征. 145 个类由原 KC1 数据集的 2 109 个模块聚合 (max、min、sum、avg) 而来; 在 94 个特征中, 其中 84 个代码特征是由原 KC1 数据集的 21 个代码特征分别按照 4 种不同聚合模式聚合而成; 此外, 聚合的类层面 KC1 数据集的 94 个特征中, 还有 10 个特征源自 C&K 度量的 10 个指标. C&K 是面向对象程序设计使用最为广泛的类层面的静态代码度量.

本实验使用的就是这个类层面 KC1 数据集, 该数据集有 95 个列, 94 个特征 (自变量), 一个响应变量 (缺陷数), 145 行关于每个类的特征数据. 在用 KC1 数据建立泊松和负二项回归中, 对所有的 $Y$ 做了加 1 处理, 以便能取对数去拟合负二项和泊松回归模型.

1) 特征选择

对 KC1 数据使用本文第 3 节的方法, 计算 MIC 进行特征选择, 最终选出了 52 个特征, 而基于泊松回归模型用 AIC 选出了 58 个特征, 用 BIC 选出了 40 个特征. 如下表 1:

**表 1　在 KC1 上各方法选到的特征**

| 所有特征 | MIC 选择的特征 (52 个) | AIC 选择的特征 (58 个) | BIC 选择的特征 (40 个) | 特征幂变换时 $\alpha$ 的取值 |
|---|---|---|---|---|
| sumLOC_TOTAL | √ | √ | | 0.45 |
| avgLOC_TOTAL | √ | | | ln |
| maxLOC_TOTAL | √ | √ | √ | 0.35 |
| minLOC_TOTAL | | √ | | 2 |
| sumNUM_UNIQUE_OPERATORS | √ | √ | √ | 0.5 |
| avgNUM_UNIQUE_OPERATORS | √ | √ | | 1.05 |
| maxNUM_UNIQUE_OPERATORS | √ | √ | √ | 1.15 |
| minNUM_UNIQUE_OPERATORS | | | | |
| sumNUM_UNIQUE_OPERANDS | √ | √ | √ | 0.5 |
| avgNUM_UNIQUE_OPERANDS | √ | √ | | 0.6 |
| maxNUM_UNIQUE_OPERANDS | √ | √ | √ | 0.65 |
| minNUM_UNIQUE_OPERANDS | | | | |
| sumNUM_OPERATORS | √ | | | 0.45 |
| avgNUM_OPERATORS | √ | √ | √ | 0.4 |
| maxNUM_OPERATORS | √ | | | 0.35 |
| minNUM_OPERATORS | | √ | √ | 0.05 |
| sumNUM_OPERANDS | | √ | √ | 0.45 |
| avgNUM_OPERANDS | | √ | √ | ln |
| maxNUM_OPERANDS | √ | | | 0.45 |
| minNUM_OPERANDS | | √ | √ | 1.95 |
| sumHALSTEAD_VOLUME | √ | | | 0.45 |
| avgHALSTEAD_VOLUME | √ | √ | | 0.4 |
| maxHALSTEAD_VOLUME | √ | √ | √ | 0.4 |
| minHALSTEAD_VOLUME | | √ | √ | 2 |
| sumHALSTEAD_PROG_TIME | √ | √ | √ | 0.35 |
| avgHALSTEAD_PROG_TIME | √ | | | ln |
| maxHALSTEAD_PROG_TIME | √ | √ | √ | 0.25 |
| minHALSTEAD_PROG_TIME | | | | |
| sumHALSTEAD_LEVEL | | √ | | ln |
| avgHALSTEAD_LEVEL | √ | | | 2 |
| maxHALSTEAD_LEVEL | | √ | | 0.05 |
| minHALSTEAD_LEVEL | | | | |
| sumHALSTEAD_LENGTH | √ | √ | √ | 0.45 |
| avgHALSTEAD_LENGTH | √ | √ | √ | 0.45 |
| maxHALSTEAD_LENGTH | √ | | | 0.4 |
| minHALSTEAD_LENGTH | | √ | √ | 0.05 |
| sumHALSTEAD_ERROR_EST | √ | | | 0.45 |
| avgHALSTEAD_ERROR_EST | √ | √ | √ | 0.45 |

**表 1　在 KC1 上各方法选到的特征 (续)**

| | | | | |
|---|---|---|---|---|
| maxHALSTEAD_ERROR_EST | √ | √ | | 0.4 |
| minHALSTEAD_ERROR_EST | | | | |
| sumHALSTEAD_EFFORT | √ | √ | √ | 0.35 |
| avgHALSTEAD_EFFORT | √ | | | 0.25 |
| maxHALSTEAD_EFFORT | √ | √ | √ | 0.25 |
| minHALSTEAD_EFFORT | | √ | | 2 |
| sumHALSTEAD_DIFFICULTY | √ | √ | √ | 0.45 |
| avgHALSTEAD_DIFFICULTY | √ | √ | | 0.75 |
| maxHALSTEAD_DIFFICULTY | √ | | | 0.75 |
| minHALSTEAD_DIFFICULTY | | √ | √ | 2 |
| sumHALSTEAD_CONTENT | √ | | | 0.55 |
| avgHALSTEAD_CONTENT | √ | | | 0.7 |
| maxHALSTEAD_CONTENT | √ | √ | | 0.55 |
| minHALSTEAD_CONTENT | | | | |
| sumLOC_EXECUTABLE | √ | √ | √ | 0.45 |
| avgLOC_EXECUTABLE | √ | √ | √ | ln |
| maxLOC_EXECUTABLE | √ | | | 0.45 |
| minLOC_EXECUTABLE | | | | |
| sumESSENTIAL_COMPLEXITY | √ | √ | √ | 0.4 |
| avgESSENTIAL_COMPLEXITY | √ | √ | | 0.05 |
| maxESSENTIAL_COMPLEXITY | | √ | | 0.05 |
| minESSENTIAL_COMPLEXITY | | | | |
| sumDESIGN_COMPLEXITY | √ | √ | √ | 0.45 |
| avgDESIGN_COMPLEXITY | √ | √ | √ | 0.05 |
| maxDESIGN_COMPLEXITY | | √ | √ | 0.05 |
| minDESIGN_COMPLEXITY | | | | |
| sumCYCLOMATIC_COMPLEXITY | √ | √ | | 0.4 |
| avgCYCLOMATIC_COMPLEXITY | √ | √ | √ | 0.05 |
| maxCYCLOMATIC_COMPLEXITY | | √ | √ | ln |
| minCYCLOMATIC_COMPLEXITY | | | | |
| sumLOC_COMMENTS | | | | |
| avgLOC_COMMENTS | | √ | | 0.2 |
| maxLOC_COMMENTS | √ | | | ln |
| minLOC_COMMENTS | | | | |
| sumLOC_CODE_AND_COMMENT | | √ | √ | 0.05 |
| avgLOC_CODE_AND_COMMENT | | √ | √ | 0.05 |
| maxLOC_CODE_AND_COMMENT | | | | |
| minLOC_CODE_AND_COMMENT | | | | |
| sumBRANCH_COUNT | √ | | | 0.4 |
| avgBRANCH_COUNT | √ | √ | √ | 0.05 |
| maxBRANCH_COUNT | √ | √ | | ln |

**表 1　在 KC1 上各方法选到的特征 (续)**

| | | | | |
|---|---|---|---|---|
| minBRANCH_COUNT | | | | |
| sumLOC_BLANK | √ | | | 0.4 |
| avgLOC_BLANK | √ | √ | √ | 0.15 |
| maxLOC_BLANK | √ | √ | √ | 0.4 |
| minLOC_BLANK | | | | |
| WEIGHTED_METHODS_PER_CLASS | | √ | √ | ln |
| RESPONSE_FOR_CLASS | | √ | | 0.35 |
| FAN_IN | | √ | | 2 |
| DEP_ON_CHILD | | √ | √ | 0.05 |
| NUM_OF_CHILDREN | | | | |
| LACK_OF_COHESION_OF_METHODS | | √ | √ | 1.3 |
| DEPTH | | | | |
| COUPLING_BETWEEN_OBJECTS | √ | √ | √ | 1.3 |
| ACCESS_TO_PUB_DATA | | | | |
| PERCENT_PUB_DATA | | √ | √ | 2 |

下面对 MIC 选择结果进行分析:

- MIC 基本上选到的是由方法聚合到类的特征, 而 AIC 选到了几乎所有关于类层面的特征;
  传统静态特征分为四类: 代码行数 (5 个)、McCabe (3 个)、Halstead (12 个)、分枝数 (1 个). 从表 1 中由 MIC 选到的 52 个特征中看出, 只有类层面的一个特征 COUPLING_BETWEEN_OBJECTS, 其余全是由 C++ 中方法层面聚合到类层面的传统的静态代码的聚合特征. 除特征 “含注释的代码行数 (LOC_CODE_AND_COMMENT)” 外, 其余 20 个传统静态代码特征, 在所使用的 4 种聚合模式中, 至少 1 种聚合所产生的特征与缺陷数有相依性. 但值得注意的是所有 min 聚合模式特征都没有被选到, 这与直觉相符合. 然而, AIC 与 BIC 却选到的几乎所有类层面的特征, 4 种聚合模式也都有选到.

- MIC 只选到三种聚合模式的特征, 而 AIC、BIC 选到的比较凌乱;
  从表 1 的结果得知, 4 种不同聚合模式应用到 21 个传统静态代码特征后产生 84 个 “新特征”. 用 MIC 选到的 avg 聚合模式特征 18个, sum 的 17 个, max 的 16 个, min 的 0 个. 因此, 从 MIC 看三种聚合模式 avg、max 和 sum 对缺陷预测都会有用. 而 AIC 和 BIC 选到的聚合特征显得比较凌乱.

2) 特征的幂次变换

采用本文 3.3 节的方法进行幂次变换, 使用 $x_i^{\alpha}$ 与 $\ln Y$ 的最大相关系数来确定每个变换的 $\alpha$. 实用中, 对每个特征做了加 1 处理, 以便可做对数变换. 我们将 $\alpha \in (0, 2]$ 的取值区

间按 0.05 间隔分为 40 个值 $(0.05, 0.1, 0.15, \ldots, 1.95, 2)$ 以及 $\ln x_i$, 从这 41 个值中选取使得相关系数最大来确定 $\alpha$, 最终确定的 $\alpha$ 在表 1 中第 3 列.

3) 主成分个数 $k$ 的确定

对数据 KC1 构造一组 $3 \times 2$ 交叉验证, 对 $k = 1, 2, 3, \ldots, 52$ 分别构建基于 PCA 的缺陷预测模型, 选取在 6 份上的 FPA 指标平均值最大的 $k$ 作为模型的主成分个数.

4) 实验结果与分析

考虑到主成分个数的选择与模型参数估计是建模中有先后的两个阶段, 因此, 选主成分个数时用了一次 $3 \times 2$ 交叉验证, 而在做回归系数估计时, 我们对数据 KC1 重新构造一组 $3 \times 2$ 交叉验证切分, 对确定的主成分个数 $k$, 在每组训练集上训练, 在测试集上测试得到结果如表 2.

以表 2 的结果实施 $m \times 2$ 交叉验证序贯 $t$-检验, 各模型与第一行的直接泊松回归有显著差异 ($p$ 值均小于 0.05), 但其他各模型之间均没有显著差异, 说明以 MIC 选出的特征构建的模型性能是不次的. 进一步, 从表 2 可以看出, 对特征做变换在多个模型上的各个性能指标上均有改善. 从指标 AAE 上看最小的是基于 BIC 选出的 40 个特征做变换的主成分负二项回归模型, 为 AAE = 4.244; 从指标 ARE 上看, 最小的是基于 MIC 选出的 52 个特征做变换的主成分泊松回归模型, 为 ARE = 1.489. 从 FPA 指标看, 最大的也是基于 MIC 选出的 52 个特征不做变换的主成分负二项回归模型, 其 FPA = 0.796. 综合各指标来看, 基于 MIC 选出的 52 个特征的模型略好于其他模型.

从表 3 结果看, 在各种情形下的泊松模型的 deviance 比负二项模型的大很多, 说明负二项回归模型的拟合效果得到了很大的改善; 似然比检验的 $G^2$ 也显著, 说明散度参数 $\theta > 0$, 这些是从训练集上模型拟合的角度来说明负二项回归模型要好.

然而软件缺陷预测模型更应该从模型的预测角度来比较模型的性能, 从表 2 的结果看, 负二项回归模型预测的性能并没有显著地比泊松回归模型好, 这说明拟合的好不见得预测的好. 传统的以模型拟合的好为目标的方法, 在软件缺陷预测问题上可能效果有限. 我们需要更加注重从预测角度建模, 模型验证与训练参数同样重要, 这也是为什么我们建议将数据一分两半, 一半做训练用、一半做验证, 重复多次, 推出 $m \times 2$ 交叉验证下的统计推断方法的原因.

从表 4 所示 PCA 的结果, 第一主成分主要由 sum 模式的聚合特征因子, 这个结果表明了求和这种聚合模式是重要的聚合方法; 而第二、第三主成分分别主要由 avg 和 max 模式聚合特征因子构成, 其他主成分很难看出其因子涵义. 同时, 我们也注意到, 由于特征未经变换, 导致选到 9 个主成分, 且无法解释 4 到 9 因子的涵义.

从表 5 可以看出, 特征经过幂次变换后, 以 FPA 为指标的主成分个数变为 2 个, 因子载荷旋转后得到的两个因子可解释性较明显, 第一个可解释为 max 和 avg 聚合特征共同作用的因子; 第二个可解释为 sum 聚合特征的因子.

对数据 KC1 上各回归模型结果的分析, 可以得出:

**表 2　KC1 数据集上各模型的性能**

| 模型 | | 评价指标 | AAE | | ARE | | FPA | |
|---|---|---|---|---|---|---|---|---|
| | | 幂次变换 | 不做变换 | 做变换 | 不做变换 | 做变换 | 不做变换 | 做变换 |
| 泊松回归 (PO) | 用 MIC 选出的 52 个特征 | 直接泊松回归 | 23.682 (5.056) | 26.113 (5.022) | 10.372 (5.370) | 10.517 (5.595) | 0.637 (0.089) | 0.729 (0.026) |
| | | 主成分泊松回归 | **4.529** (**0.555**) ($\boldsymbol{k=1}$) | 4.486 (0.464) ($k=2$) | **1.706** (**0.346**) ($\boldsymbol{k=9}$) | **1.489** (**0.300**) ($\boldsymbol{k=10}$) | 0.783 (0.034) ($k=9$) | 0.789 (0.035) ($k=2$) |
| | 用 AIC* 选出的 58 个特征 | 主成分泊松回归 | 4.615 (0.499) ($k=2$) | 4.531 (0.452) ($k=3$) | 1.848 (0.758) ($k=13$) | 1.683 (0.377) ($k=14$) | **0.794** (**0.033**) ($\boldsymbol{k=12}$) | 0.792 (0.039) ($k=4$) |
| | 用 BIC* 选出的 40 个特征 | 主成分泊松回归 | 4.576 (0.511) ($k=2$) | **4.336** (**0.488**) ($\boldsymbol{k=2}$) | 1.943 (0.280) ($k=1$) | 1.639 (0.257) ($k=2$) | 0.780 (0.048) ($k=10$) | 0.790 (0.038) ($k=2$) |
| | 用全部 86* 个特征 | 主成分泊松回归 | 4.614 (0.638) ($k=2$) | 4.480 (0.659) ($k=2$) | 1.731 (0.667) ($k=13$) | 1.643 (0.354) ($k=2$) | 0.780 (0.039) ($k=13$) | **0.793** (**0.038**) ($\boldsymbol{k=4}$) |
| 负二项回归 (NB2) | 用 MIC 选出的 52 个特征 | 主成分负二项回归 | 4.766 (1.106) ($k=1$) ($\theta=1.274$) | 4.342 (0.757) ($k=8$) ($\theta=2.188$) | **1.637** (**0.250**) ($\boldsymbol{k=5}$) ($\boldsymbol{\theta=1.762}$) | **1.552** (**0.270**) ($\boldsymbol{k=7}$) ($\boldsymbol{\theta=2.118}$) | **0.796** (**0.032**) ($\boldsymbol{k=9}$) ($\boldsymbol{\theta=2.242}$) | 0.789 (0.033) ($k=2$) ($\theta=1.624$) |
| | 用 AIC* 选出的 58 个特征 | 主成分负二项回归 | 4.753 (0.934) ($k=1$) ($\theta=1.265$) | 4.394 (0.483) ($k=4$) ($\theta=1.726$) | 1.688 (0.558) ($k=13$) ($\theta=2.480$) | 1.692 (0.312) ($k=12$) ($\theta=2.271$) | 0.794 (0.033) ($k=12$) ($\theta=2.390$) | 0.791 (0.038) ($k=4$) ($\theta=1.726$) |
| | 用 BIC* 选出的 40 个特征 | 主成分负二项回归 | 5.059 (1.326) ($k=1$) ($\theta=1.262$) | **4.244** (**0.521**) ($\boldsymbol{k=2}$) ($\boldsymbol{\theta=1.611}$) | 1.918 (0.416) ($k=1$) ($\theta=1.262$) | 1.627 (0.302) ($k=2$) ($\theta=1.611$) | 0.783 (0.043) ($k=10$) ($\theta=2.451$) | 0.791 (0.040) ($k=2$) ($\theta=1.611$) |
| | 用全部 86* 个特征 | 主成分负二项回归 | **4.695** (**0.929**) ($\boldsymbol{k=1}$) ($\boldsymbol{\theta=1.267}$) | 4.366 (0.544) ($k=3$) ($\theta=1.694$) | 1.748 (0.768) ($k=13$) ($\theta=2.551$) | 1.700 (0.328) ($k=14$) ($\theta=2.495$) | 0.780 (0.037) ($k=16$) ($\theta=3.820$) | **0.795** (**0.037**) ($\boldsymbol{k=4}$) ($\boldsymbol{\theta=1.715}$) |

* 注: 1) AIC 的 58, BIC 的 40 个特征是以泊松回归模型选出的, 为了便于比较在负二项时没有重新选;
2) KC1 中有 94 个属性, 删除 8 个常量, 留下 86 个属性.

1) 使用 sum、avg、max 聚合模式得到的聚合特征对模型的预测性能的影响是显著的: 从表 6 看出, 以 MIC 选到的 52 个特征中的最好模型为 Baseline, 在这个模型中去掉第一主成分, 得到的模型与 Baseline 比较, 在 $T_{m\times 2}$ 检验下都有显著差异 (除了 NB2 的 AAE 指标). 说明两个第一主成分对缺陷预测都是有显著作用的. 而在特征不做变换情形下, 第一主成分对应的是 sum 类型的因子; 在特征做变换下, 第一主成分对应的是 max 和 avg 类型因子. 这说明聚合特征 (sum、avg、max) 对预测性能的影响是显著的, 不同的聚合模式

**表 3 散度参数的检验**

| 性能指标 | 模型 | | 似然比检验 $G^2$ | | |
|---|---|---|---|---|---|
| | | | 泊松回归 deviance | 负二项回归 deviance | $G^2$ |
| AAE | MIC 52 个特征 | 不做变换 | 366.136 | 67.981 | 298.155* |
| | | 做变换 | 301.360 | 63.989 | 237.371* |
| | AIC 58 个特征 | 不做变换 | 371.750 | 68.068 | 303.682* |
| | | 做变换 | 295.861 | 63.955 | 231.906* |
| | BIC 40 个特征 | 不做变换 | 369.994 | 68.179 | 301.815* |
| | | 做变换 | 306.935 | 63.916 | 243.019* |
| ARE | MIC 52 个特征 | 不做变换 | 265.552 | 63.096 | 202.456* |
| | | 做变换 | 238.766 | 63.011 | 175.755* |
| | AIC 58 个特征 | 不做变换 | 200.602 | 60.210 | 140.392* |
| | | 做变换 | 221.305 | 60.913 | 160.392* |
| | BIC 40 个特征 | 不做变换 | 369.994 | 68.179 | 301.815* |
| | | 做变换 | 306.935 | 63.916 | 243.019* |
| FPA | MIC 52 个特征 | 不做变换 | 224.070 | 61.798 | 162.272* |
| | | 做变换 | 301.360 | 63.989 | 237.371* |
| | AIC 58 个特征 | 不做变换 | 206.129 | 61.117 | 145.012* |
| | | 做变换 | 287.981 | 63.967 | 224.013* |
| | BIC 40 个特征 | 不做变换 | 212.635 | 62.008 | 150.627* |
| | | 做变换 | 306.936 | 63.916 | 243.019* |

注: 表中 * 代表在 1% 显著水平下显著.

对不同模型性能的影响是有差异的.

2) 模型的可解释性较好: 从表 4 也可以看出来, 对 52 个特征不做变换可以得到 9 个主成分, 因子载荷旋转后的可解释性非常明显, 第一个可解释为 sum 模式的因子, 即, 主要从 sum 聚合模式的特征中获得; 第二个可解释为 avg 模式的因子, 即主要从 avg 聚合模式的特征中获得; 第三个可解释为 max 模式的因子, 即主要从 max 聚合模式的特征中获得. 但是剩余的其他 6 个因子没有明显的类型特征, 不好解释. 从表 5 可以看出, 特征经过幂次变换后, 主成分个数变为 2 个, 因子载荷旋转后得到的两个因子可解释性较明显, 第一个可解释为 max 和 avg 模式的特征共同作用的因子, 即主要从 max 和 avg 聚合模式的特征集中获得; 第二个可解释为 sum 模式的因子, 即主要从 sum 聚合模式的特征中获得.

3) 对特征做变换比不做变换要好: 从表 2 可以看出: 除了基于 AIC 的变换后的主成分负二项回归模型的 ARE 指标外, 变换后的 AAE、ARE 指标值绝大多数都低于不做变换时的值, 多个模型下的 FPA 指标高于不做变换时的值, 许多模型下的指标均有改善, 说明对特征做变换是有作用的. 主要是变换后的特征与因变量 $\ln Y$ 之间更加接近线性关系, 且特征的幂变换的系数 $\alpha$ 绝大部分小于 1, 压缩了聚合后的特征中取值较大的观测值, 从统计意义上来说, 就是特征的取值更加稳定了, 方差变小了, 使得特征变换后主成分个数从 9 个降到 2 个, 模型变得更为简洁, 性能还略有提升且更加稳定, 这说明变换是有效的.

**表 4 使用 52 个特征做主成分 (不做变换) 的旋转后的 9 个因子载荷**

| | 因子 1 | 因子 2 | 因子 3 | 因子 4 | 因子 5 | 因子 6 | 因子 7 | 因子 8 | 因子 9 |
|---|---|---|---|---|---|---|---|---|---|
| sumLOC_TOTAL | 0.856 | 0.322 | 0.275 | 0.105 | 0.105 | 0.076 | 0.184 | -0.054 | 0.000 |
| sumLOC_EXECUTABLE | 0.843 | 0.345 | 0.299 | 0.098 | 0.108 | 0.062 | 0.165 | -0.060 | -0.045 |
| maxHALSTEAD_VOLUME | 0.353 | 0.384 | 0.841 | 0.102 | 0.027 | 0.035 | 0.049 | 0.019 | 0.030 |
| avgHALSTEAD_VOLUME | 0.290 | 0.851 | 0.388 | 0.130 | -0.041 | 0.067 | 0.064 | 0.034 | 0.080 |
| maxNUM_OPERANDS | 0.361 | 0.409 | 0.816 | 0.113 | 0.046 | 0.038 | 0.070 | 0.072 | 0.041 |
| maxHALSTEAD_LENGTH | 0.352 | 0.411 | 0.823 | 0.113 | 0.075 | 0.037 | 0.059 | 0.043 | 0.025 |
| maxNUM_OPERATORS | 0.353 | 0.408 | 0.822 | 0.112 | 0.092 | 0.037 | 0.049 | 0.029 | 0.017 |
| sumHALSTEAD_DIFFICULTY | 0.951 | 0.159 | 0.218 | 0.014 | 0.053 | 0.045 | 0.018 | 0.112 | 0.000 |
| maxLOC_TOTAL | 0.319 | 0.497 | 0.715 | 0.050 | 0.170 | 0.060 | 0.269 | -0.014 | 0.120 |
| sumHALSTEAD_LENGTH | 0.920 | 0.268 | 0.262 | 0.066 | 0.013 | 0.043 | 0.051 | 0.014 | 0.026 |
| avgHALSTEAD_LENGTH | 0.303 | 0.858 | 0.357 | 0.155 | 0.004 | 0.082 | 0.067 | 0.051 | 0.039 |
| maxHALSTEAD_ERROR_EST | 0.352 | 0.385 | 0.841 | 0.103 | 0.027 | 0.036 | 0.048 | 0.019 | 0.030 |
| sumNUM_UNIQUE_OPERANDS | 0.916 | 0.244 | 0.220 | 0.169 | 0.040 | 0.048 | 0.079 | 0.065 | 0.033 |
| maxHALSTEAD_CONTENT | 0.318 | 0.366 | 0.567 | 0.576 | 0.173 | -0.013 | 0.013 | 0.002 | 0.100 |
| avgHALSTEAD_ERROR_EST | 0.289 | 0.852 | 0.385 | 0.129 | -0.039 | 0.076 | 0.063 | 0.035 | 0.079 |
| sumHALSTEAD_VOLUME | 0.908 | 0.292 | 0.271 | 0.049 | -0.026 | 0.036 | 0.040 | 0.007 | 0.060 |
| maxLOC_EXECUTABLE | 0.352 | 0.470 | 0.748 | 0.049 | 0.184 | 0.043 | 0.184 | 0.005 | 0.029 |
| avgLOC_TOTAL | 0.218 | 0.872 | 0.267 | 0.091 | 0.082 | 0.164 | 0.180 | -0.041 | 0.115 |
| sumHALSTEAD_EFFORT | 0.884 | 0.253 | 0.287 | -0.085 | -0.137 | 0.011 | -0.050 | 0.043 | 0.127 |
| sumHALSTEAD_PROG_TIME | 0.884 | 0.253 | 0.287 | -0.085 | -0.137 | 0.011 | -0.050 | 0.043 | 0.127 |
| avgNUM_OPERATORS | 0.298 | 0.863 | 0.356 | 0.140 | 0.025 | 0.083 | 0.057 | 0.039 | 0.019 |
| maxNUM_UNIQUE_OPERANDS | 0.328 | 0.461 | 0.705 | 0.305 | 0.102 | 0.030 | 0.055 | 0.162 | 0.129 |
| sumNUM_OPERATORS | 0.917 | 0.267 | 0.272 | 0.060 | 0.028 | 0.044 | 0.053 | 0.004 | 0.009 |
| maxLOC_BLANK | 0.515 | 0.484 | 0.350 | 0.017 | 0.082 | -0.007 | 0.159 | 0.116 | 0.538 |
| maxHALSTEAD_EFFORT | 0.402 | 0.300 | 0.842 | -0.082 | -0.110 | 0.026 | 0.018 | 0.038 | 0.005 |
| maxHALSTEAD_PROG_TIME | 0.402 | 0.300 | 0.842 | -0.082 | -0.110 | 0.026 | 0.018 | 0.038 | 0.005 |
| avgNUM_UNIQUE_OPERANDS | 0.283 | 0.831 | 0.271 | 0.319 | 0.008 | 0.108 | 0.090 | 0.125 | 0.036 |
| sumHALSTEAD_ERROR_EST | 0.907 | 0.295 | 0.271 | 0.047 | -0.027 | 0.038 | 0.041 | 0.006 | 0.059 |
| avgHALSTEAD_EFFORT | 0.274 | 0.747 | 0.516 | -0.078 | -0.179 | -0.009 | 0.018 | 0.074 | 0.156 |
| avgHALSTEAD_PROG_TIME | 0.274 | 0.747 | 0.516 | -0.078 | -0.179 | -0.009 | 0.018 | 0.074 | 0.156 |
| avgHALSTEAD_DIFFICULTY | 0.364 | 0.814 | 0.293 | 0.054 | 0.081 | 0.104 | 0.059 | 0.254 | -0.053 |
| sumHALSTEAD_CONTENT | 0.870 | 0.173 | 0.221 | 0.284 | 0.140 | 0.049 | 0.119 | 0.043 | -0.058 |
| avgBRANCH_COUNT | 0.265 | 0.878 | 0.320 | -0.124 | 0.138 | 0.028 | 0.015 | -0.006 | -0.007 |
| sumNUM_UNIQUE_OPERATORS | 0.930 | 0.100 | 0.216 | 0.083 | 0.117 | 0.050 | 0.054 | 0.162 | -0.033 |
| maxHALSTEAD_DIFFICULTY | 0.361 | 0.465 | 0.614 | -0.013 | 0.219 | 0.026 | 0.087 | 0.411 | 0.069 |
| avgHALSTEAD_CONTENT | 0.260 | 0.749 | 0.211 | 0.493 | 0.084 | 0.073 | 0.095 | 0.081 | -0.066 |
| avgLOC_EXECUTABLE | 0.224 | 0.893 | 0.295 | 0.109 | 0.105 | 0.106 | 0.124 | 0.006 | 0.028 |
| sumDESIGN_COMPLEXITY | 0.899 | 0.237 | 0.301 | 0.005 | 0.156 | 0.046 | 0.064 | -0.013 | -0.059 |
| sumBRANCH_COUNT | 0.917 | 0.270 | 0.263 | -0.055 | 0.066 | 0.029 | 0.024 | 0.018 | 0.018 |

**表 4 使用 52 个特征做主成分 (不做变换) 的旋转后的 9 个因子载荷 (续)**

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| avgCYCLOMATIC_COMPLEXITY | 0.268 | 0.878 | 0.317 | -0.122 | 0.145 | 0.031 | 0.019 | -0.016 | -0.007 |
| sumCYCLOMATIC_COMPLEXITY | 0.926 | 0.233 | 0.264 | -0.026 | 0.094 | 0.030 | 0.033 | 0.033 | 0.008 |
| avgDESIGN_COMPLEXITY | 0.243 | 0.868 | 0.311 | -0.087 | 0.218 | 0.065 | -0.006 | -0.072 | -0.044 |
| sumLOC_BLANK | 0.872 | 0.320 | 0.119 | 0.048 | -0.015 | 0.001 | 0.053 | 0.017 | 0.311 |
| avgNUM_UNIQUE_OPERATORS | 0.333 | 0.770 | 0.222 | 0.170 | 0.123 | 0.112 | 0.070 | 0.349 | -0.109 |
| avgLOC_BLANK | 0.268 | 0.755 | 0.169 | 0.084 | 0.040 | 0.052 | 0.164 | -0.011 | 0.480 |
| COUPLING_BETWEEN_OBJECTS | 0.257 | 0.476 | 0.298 | 0.155 | 0.646 | 0.093 | 0.121 | 0.190 | 0.052 |
| sumESSENTIAL_COMPLEXITY | 0.925 | 0.203 | 0.223 | -0.029 | 0.053 | -0.007 | -0.029 | 0.093 | 0.076 |
| maxNUM_UNIQUE_OPERATORS | 0.225 | 0.537 | 0.492 | 0.055 | 0.238 | 0.044 | 0.082 | 0.556 | 0.103 |
| avgHALSTEAD_LEVEL | -0.156 | -0.302 | -0.073 | -0.016 | -0.040 | -0.930 | -0.012 | -0.022 | -0.006 |
| avgESSENTIAL_COMPLEXITY | 0.238 | 0.815 | 0.282 | -0.240 | -0.012 | -0.114 | -0.018 | 0.004 | 0.075 |
| maxBRANCH_COUNT | 0.378 | 0.466 | 0.715 | -0.123 | 0.229 | 0.001 | 0.122 | 0.039 | -0.027 |
| maxLOC_COMMENTS | 0.244 | 0.395 | 0.351 | 0.036 | 0.077 | 0.015 | 0.787 | 0.055 | 0.085 |

4) 基于 MIC 特征选择方法的回归模型的性能较好: 从指标 AAE 上看, 最小的是基于 BIC 选出的 40 个特征做变换的主成分负二项回归模型, AAE = 4.244. 从指标 ARE 上看, 最小的是基于 MIC 选出的 52 个特征做变换的主成分泊松回归模型, ARE = 1.489. 从 FPA 指标看, 最大的是基于 MIC 选出的 52 个特征不做变换时的主成分负二项回归模型, 其 FPA = 0.796. 单看负二项回归模型, 三个指标中, 最好的还是基于 MIC 选出的 52 个特征的负二项回归模型. 虽然, 从统计的显著性对比来看, 基于 MIC 的模型, 与基于 AIC、BIC 以及 86 个特征的模型的性能均没有显著差异, 但多项指标都略好于其他模型, 综合来说, 可以得出基于 MIC 选出的特征建模是有优势的.

## §7. 总结与展望

针对基于静态代码的软件缺陷预测建模中特征多、特征间关系复杂, 而传统的特征选择得到的特征集可解释性差的问题, 本文先采用 Reshef 等 [4] 提出的 MIC 度量从众多特征中预选出 MIC 较大的候选特征集, 其次对所选特征进行了适当的幂次变换, 然后再对变换后的特征集进行主成分分析, 最后基于得到的主成分做泊松和负二项回归模型. 实验结果表明, 基于 MIC 特征选择方法得到的回归模型性能略好于传统的基于 AIC、BIC 方法, 且进一步因子分析表明, 得到的前两个主成分一个是对应到 avg 和 max 聚合模式特征集, 一个正好对应到 sum 聚合模式特征集, 使得模型具有良好的可解释性. 下一步, 我们将研究其他聚合模式 (比如, 分位数) 对缺陷预测是否有显著作用, 寻找更多有用的聚合模式; 另外, 既然聚合特征是由类中的方法模块聚合得到, 是否可以考虑聚合后数据与聚合前数据的层次集成学习模型, 以进一步提高软件缺陷预测的性能.

**表 5 使用 52 个特征做变换后做主成分分析经旋转后的 2 个因子载荷**

| | 因子 1 | 因子 2 |
|---|---|---|
| sumLOC_TOTAL | 0.484 | 0.846 |
| sumLOC_EXECUTABLE | 0.577 | 0.799 |
| maxHALSTEAD_VOLUME | 0.808 | 0.522 |
| avgHALSTEAD_VOLUME | 0.892 | 0.405 |
| maxNUM_OPERANDS | 0.803 | 0.53 |
| maxHALSTEAD_LENGTH | 0.812 | 0.521 |
| maxNUM_OPERATORS | 0.815 | 0.514 |
| sumHALSTEAD_DIFFICULTY | 0.415 | 0.895 |
| maxLOC_TOTAL | 0.818 | 0.485 |
| sumHALSTEAD_LENGTH | 0.508 | 0.855 |
| avgHALSTEAD_LENGTH | 0.879 | 0.416 |
| maxHALSTEAD_ERROR_EST | 0.81 | 0.518 |
| sumNUM_UNIQUE_OPERANDS | 0.434 | 0.887 |
| maxHALSTEAD_CONTENT | 0.667 | 0.547 |
| avgHALSTEAD_ERROR_EST | 0.891 | 0.406 |
| sumHALSTEAD_VOLUME | 0.545 | 0.827 |
| maxLOC_EXECUTABLE | 0.829 | 0.499 |
| avgLOC_TOTAL | 0.841 | 0.348 |
| sumHALSTEAD_EFFORT | 0.66 | 0.727 |
| sumHALSTEAD_PROG_TIME | 0.66 | 0.727 |
| avgNUM_OPERATORS | 0.879 | 0.411 |
| maxNUM_UNIQUE_OPERANDS | 0.781 | 0.529 |
| sumNUM_OPERATORS | 0.506 | 0.856 |
| maxLOC_BLANK | 0.765 | 0.487 |
| maxHALSTEAD_EFFORT | 0.826 | 0.503 |
| maxHALSTEAD_PROG_TIME | 0.826 | 0.503 |
| avgNUM_UNIQUE_OPERANDS | 0.838 | 0.422 |
| sumHALSTEAD_ERROR_EST | 0.561 | 0.816 |
| avgHALSTEAD_EFFORT | 0.913 | 0.385 |
| avgHALSTEAD_PROG_TIME | 0.905 | 0.38 |
| avgHALSTEAD_DIFFICULTY | 0.827 | 0.444 |
| sumHALSTEAD_CONTENT | 0.327 | 0.919 |
| avgBRANCH_COUNT | 0.879 | 0.343 |
| sumNUM_UNIQUE_OPERATORS | 0.312 | 0.929 |
| maxHALSTEAD_DIFFICULTY | 0.776 | 0.513 |
| avgHALSTEAD_CONTENT | 0.748 | 0.44 |
| avgLOC_EXECUTABLE | 0.885 | 0.364 |
| sumDESIGN_COMPLEXITY | 0.41 | 0.894 |
| sumBRANCH_COUNT | 0.496 | 0.846 |
| avgCYCLOMATIC_COMPLEXITY | 0.87 | 0.343 |

**表 5 使用 52 个特征做变换后做主成分分析经旋转后的 2 个因子载荷 (续)**

| | | |
|---|---|---|
| sumCYCLOMATIC_COMPLEXITY | 0.437 | 0.882 |
| avgDESIGN_COMPLEXITY | 0.856 | 0.34 |
| sumLOC_BLANK | 0.576 | 0.743 |
| avgNUM_UNIQUE_OPERATORS | 0.783 | 0.457 |
| avgLOC_BLANK | 0.787 | 0.383 |
| COUPLING_BETWEEN_OBJECTS | 0.6 | 0.5 |
| sumESSENTIAL_COMPLEXITY | 0.342 | 0.905 |
| maxNUM_UNIQUE_OPERATORS | 0.817 | 0.404 |
| avgHALSTEAD_LEVEL | -0.41 | -0.251 |
| avgESSENTIAL_COMPLEXITY | 0.738 | 0.318 |
| maxBRANCH_COUNT | 0.815 | 0.471 |
| maxLOC_COMMENTS | 0.646 | 0.45 |

**表 6 KC1 数据集上特征的第一主成分的作用**

| | 评价指标 | AAE | | ARE | | FPA | |
|---|---|---|---|---|---|---|---|
| | | 不做变换 | 做变换 | 不做变换 | 做变换 | 不做变换 | 做变换 |
| 泊松回归模型 (PO) | 52 个特征 (MIC) 做主成分泊松回归 (Baseline) | **4.529** (**0.555**) | **4.486** (**0.464**) | **1.706** (**0.346**) | **1.489** (**0.300**) | **0.783** (**0.034**) | **0.789** (**0.035**) |
| | 52 个特征 (MIC) 去掉第一个主成分做的泊松回归 | **5.702** (**0.482**) | **5.666** (**0.468**) | **2.663** (**0.113**) | **2.561** (**0.212**) | **0.529** (**0.101**) | **0.668** (**0.063**) |
| | 与 Baseline 差异的 $m\times 2$ CV $t$-检验 | $T_{4\times 2}$:**2.301** $p$:**0.027** | $T_{3\times 2}$:**2.700** $p$:**0.021** | $T_{3\times 2}$:**2.889** $p$:**0.017** | $T_{3\times 2}$:**2.358** $p$:**0.032** | $T_{4\times 2}$:**-2.343** $p$:**0.026** | $T_{4\times 2}$:**-3.119** $p$:**0.008** |
| 负二项回归模型 (NB2) | 52 个特征 (MIC) 做主成分负二项回归 (Baseline) | 4.766 (1.106) | 4.342 (0.757) | **1.637** (**0.250**) | **1.552** (**0.270**) | **0.796** (**0.032**) | **0.789** (**0.033**) |
| | 52 个特征 (MIC) 去掉第一个主成分做的负二项回归 | 5.809 (0.494) | 5.784 (0.433) | **2.738** (**0.231**) | **2.749** (**0.416**) | **0.561** (**0.077**) | **0.668** (**0.063**) |
| | 与 Baseline 差异的 $m\times 2$ CV $t$-检验 | $T_{3\times 2}$:0.729 $p$:0.249 | $T_{3\times 2}$:1.675 $p$:0.077 | $T_{3\times 2}$:**3.334** $p$:**0.010** | $T_{3\times 2}$:**2.365** $p$:**0.032** | $T_{3\times 2}$:**-3.038** $p$:**0.014** | $T_{4\times 2}$:**-3.041** $p$:**0.009** |

注: 黑体代表水平 0.05 下显著. AAE 指标下, 负二项主成分回归模型下, 去掉与不去掉第一主成分两模型的差, 当持续增加 $m$ 序贯检验没有改善, 故表中只留下 $3\times 2$ 的结果.

## 参 考 文 献

[1] KHOSHGOFTAAR T M, ALLEN E B, XU Z W. Predicting testability of program modules using a neural network [C] // *Proceedings 3rd IEEE Symposium on Application-Specific Systems and Software Engineering Technology*. Los Alamitos, CA: IEEE Computer Society, 2000: 57–62.

[2] KHOSHGOFTAAR T M, SELIYA N. Tree-based software quality estimation models for fault prediction [C] // *Proceedings Eighth IEEE Symposium on Software Metrics*. Los Alamitos, CA: IEEE Computer Society, 2002: 203–214.

[3] NIKORA A P, MUNSON J C. Developing fault predictors for evolving software systems [C] // *Proceedings Ninth International Software Metrics Symposium*. Los Alamitos, CA: IEEE Computer Society, 2003: 338–350.

[4] RESHEF D N, RESHEF Y A, FINUCANE H K, et al. Detecting novel associations in large data sets [J]. *Science*, 2011, **334(6062)**: 1518–1524.

[5] TANTITHAMTHAVORN C, MCINTOSH S, HASSAN A E, et al. An empirical comparison of model validation techniques for defect prediction models [J]. *IEEE Trans Software Engrg*, 2017, **43(1)**: 1–18.

[6] MA Y, GUO L, CUKIC B. A statistical framework for the prediction of fault-proneness [M] // ZHANG D, TSAI J J P. *Advances in Machine Learning Applications in Software Engineering*. Hershey PA: Idea Group, 2007: 237–263.

[7] GUO L, MA Y, CUKIC B, et al. Robust prediction of fault-proneness by random forests [C] // *Proceedings 15th International Symposium on Software Reliability Engineering*. Los Alamitos, CA: IEEE Computer Society, 2004: 417–428.

[8] KHOSHGOFTAAR T M, GAO K H, NAPOLITANO A. An empirical study of feature ranking techniques for software quality prediction [J]. *Int J Softw Eng Know*, 2012, **22(2)**:161–183.

[9] OKUTAN A, YILDIZ O T. Software defect prediction using Bayesian networks [J]. *Empir Softw Eng*, 2014, **19(1)**: 154–181.

[10] GIL Y, LALOUCHE G. On the correlation between size and metric validity [J]. *Empir Softw Eng*, 2017, **22(5)**: 2585–2611.

[11] GAO K H, KHOSHGOFTAAR T M. A comprehensive empirical study of count models for software fault prediction [J]. *IEEE Trans Reliab*, 2007, **56(2)**: 223–236.

[12] 陈翔, 顾庆, 刘望舒, 等. 静态软件缺陷预测方法研究 [J]. 软件学报, 2016, **27(1)**: 1–25.

[13] 王青, 伍书剑, 李明树. 软件缺陷预测技术 [J]. 软件学报, 2008, **19(7)**: 1565–1580.

[14] KORU A G, LIU H D. An investigation of the effect of module size on defect prediction using static measures [C] // *Proceedings of the 2005 Workshop on Predictor Models in Software Engineering, PROMISE 2005*. New York: Association for Computing Machinery, 2005: 1–5.

[15] ZHANG F, HASSAN A E, MCINTOSH S, et al. The use of summation to aggregate software metrics hinders the performance of defect prediction models [J]. *IEEE Trans Software Engrg*, 2017, **43(5)**: 476–491.

[16] OSTRAND T J, WEYUKER E J, BELL R M. Predicting the location and number of faults in large software systems [J]. *IEEE Trans Software Engrg*, 2005, **31(4)**: 340–355.

[17] AKAIKE H. Information theory and an extension of the maximum likelihood principle [C] // PETROV B N, CSÁDKI F (eds). *Second International Symposium on Information Theory, Tsahkadsor, Armenia, USSR, September 2-8, 1971*, Budapest: Akadémiai Kiadó, 1973: 267–281.

[18] SCHWARZ G. Estimating the dimension of a model [J], *Ann Statist*, 1978, **6(2)**: 461–464.

[19] XU Z, XUAN J F, LIU J, et al. MICHAC: Defect prediction via feature selection based on maximal information coefficient with hierarchical agglomerative clustering [C] // *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering*, Los Alamitos, CA: IEEE Computer Society, 2016: 370–381.

[20] HILBE J M. Negative Binomial Regression [M]. Cambridge, UK: Cambridge University Press, 2011.

[21] WANG Y, WANG R B, JIA H C, et al. Blocked $3 \times 2$ cross-validated $t$-test for comparing supervised classification learning algorithms [J]. *Neural Comput*, 2014, **26(1)**: 208–235.

[22] WANG R B, WANG Y, LI J H, et al. Block-regularized $m \times 2$ cross-validated estimator of the generalization error [J]. *Neural Comput*, 2017, **29(2)**: 519–554.

[23] WANG R B, LI J H, YANG X L. Block-regularized $m \times 2$ cross-validated sequential $t$-test for comparing two supervised machine learning algorithms [J]. *Neural Comput*, 2018, Submited.

# Software Defect Prediction Model Based on Maximal Information Coefficient

CUI Jun LIU Yana

(*School of Computer Information and Technology, Shanxi University, Taiyuan, 030006, China*)

GUO Xinfeng WANG Ruibo LI Jihong

(*School of Software Engineering, Shanxi University, Taiyuan, 030002, China*)

**Abstract:** In software defect prediction with a regression model, too many metrics extracted from static code and aggregated (sum, avg, max, min) from methods into classes can be candidate features, and the classical feature selection methods, such as AIC, BIC, should be processed at a given model. As a result, the selected feature sets are significantly different for various models without a reasonable interpretation. Maximal information coefficient (MIC) presented by Reshef et al. [4] is a novel method to measure the degree of the interdependence between two continuous variables, and an available computing method is also given based on the observations. This paper firstly use the MIC between defect counts and each feature to select features, and then conduct the power transformation on the selected features, and finally build up the principal component Poisson and negative binomial regression model. All experiments are conducted on KC1 data set in NASA repository on the level of class. The block-regularized $m \times 2$ cross-validated sequential $t$-test is employed to test the difference of performance of two models. The performance measures of a model in this paper are FPA, AAE, ARE. The experimental results show that 1) the aggregated features, such as sum, avg, max, are selected by MIC except min, which are significantly different from AIC, BIC; 2) the power transformation to the features can improve the performance for majority of models; 3) after PCA and factorial analysis, two clear factors are obtained in the model. One corresponds to the aggregated features via avg and max, and the other corresponds to the aggregated features with sum. Therefore, the model owns a reasonable interpretation. Conclusively, the aggregated features with sum, avg, max are significantly effective for software defect prediction, and the regression model based on the selected features by MIC has some advantages.

**Keywords:** MIC; software defect prediction; Possion regression; negative binomial regression; block-regularized $m \times 2$ cross-validated sequential $t$-test

**2010 Mathematics Subject Classification:** 62P30