



貴州大學
GUIZHOU UNIVERSITY

主标题

副标题

姓名 _____

学号 _____

教师 _____

院所 _____

19th August 2024

Contents

| | | |
|----------|--------------------|-----------|
| 1 | 背景 | 3 |
| 2 | 问题一 | 4 |
| 2.1 | 寻找特征 | 4 |
| 2.2 | 提取特征 | 5 |
| 3 | 问题二模型的建立与求解 | 7 |
| 3.1 | 数据分析 | 7 |
| 3.2 | 模型说明 | 12 |
| 3.2.1 | Gini 系数的计算 | 12 |
| 3.2.2 | 连续属性划分点 t 的确定 | 12 |
| 3.2.3 | 处理连续属性 | 13 |
| 3.2.4 | 处理缺失值 | 13 |
| 3.2.5 | 选择最优划分 | 13 |
| 3.2.6 | 构建树结构 | 13 |
| 3.3 | 模型建立 | 14 |
| 4 | 问题 3 | 15 |
| 4.1 | 决策树模型分析 | 15 |
| 4.2 | 模型改进 | 18 |
| 5 | 问题四 | 22 |
| 5.1 | 自动特征提取 | 22 |
| 5.2 | 增加数据集多样性 | 22 |
| 5.3 | 结合静态和动态特征 | 22 |

1 背景

电子计算机自诞生以来,经历了快速而显著的发展。在短短的百年时间里,计算机在体积、能耗、计算速度和应用能力等方面都发生了巨大变化。然而,要充分发挥计算机的潜力,必须使用能够被其解释和执行的指令序列,即程序。最初,这些程序是通过机器语言编写的,但由于其不直观,极大地限制了计算机的普及。为了解决这一问题,1957 年,第一个自动编译器 FORTRAN 诞生了。此后,许多性能更高且支持接近自然语言的编译器相继被设计出来,如 C/C++ 编译器 GCC, Clang。编译器的出现极大地推动了计算机在现代社会的广泛应用。

为了便于使用计算机,人们首先需要按照特定规则(即程序设计语言,例如 C++ 语言)将待执行的指令以特定顺序编写成源代码。这些源代码随后通过编译器自动翻译(即编译)为一系列机器语言指令(即汇编语言),最后通过链接器各种库文件(链接库),生成可执行程序,并提交给计算机执行。源代码是程序的初始形式,通常以高级编程语言编写,具有良好的可维护性,并且形式上更加接近自然语言。而汇编语言则是一种机器语言,本质上是由 01 一一序列映射成的低级程序设计语言,如图 1 所示,一条汇编语言语句通常由操作码,操作数组成,操作码提供本条指令要执行的操作,而操作数提供数据的来源,图 1 中展示的汇编语句的作用是将寄存器 `rdx` 和 `rax` 中的数据取出来相加,然后把结果放回到 `rdx` 寄存器中。

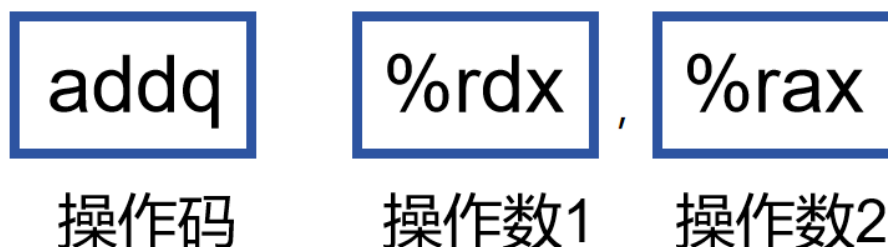


Figure 1: 汇编语句的组成

编译器则是将源代码翻译为机器语言的工具,汇编语言作为机器语言的中间形式,提供了一种更易于理解和操作的方式。编译器的工作流程如图 2 所示。编译器会根据源代码构建一个语法树,然后根据语法树的规则转翻译源代码为汇编语言代码。不同版本的编译器在转译时有部分不同的处理方式,例如通过指令调度算法来减少跳转,寄存器换名技术来消除数据冲突。随着编程语言的演变,编译器也在不断更新。例如, GCC (GNU Compiler Collection) 已经更新到 13.2.0 版本。不同版本的编译器在编译同一程序时可能产生不同的编译结果;即使是相同版本的编译器,使用不同的编译选项也会导致编译结果的差异。因此,研究如何利用这些差异来区分编译器的版本具有重要意义。

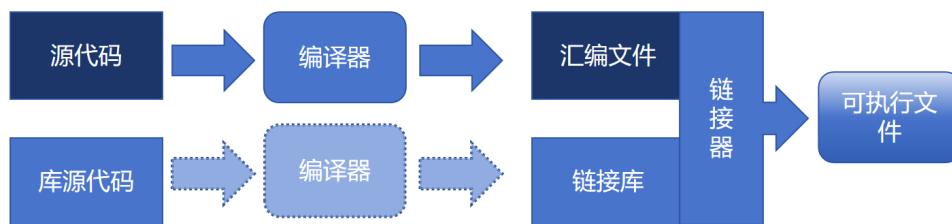


Figure 2: 编译器的工作流程

2 问题一

2.1 寻找特征

根据附件提供的编译器版本信息，我们综合考量了各版本的使用情况和差异程度，选择了以下五个 GCC 版本作为识别对象：“8.4.0”、“10.2.0”、“11.3.0”、“12.2.0”和“13.2.0”。将程序源代码 1 分别由这五个版本的 GCC 编译器进行编译，生成对应的五个汇编文件。我们利用 VS Code 自带的文本比较工具，对这五个汇编文件进行两两比对。VS Code 的文本比较工具主要是基于 ‘Myers diff’ 算法实现，该算法是由 Eugene Myers 在 1986 年提出的一种广泛使用的最短编辑距离算法，专门用于计算两个序列（例如文本文件的行或字符）之间的最小修改步骤（插入、删除、替换）。

Table 1: 8.4.0 与 10.2.0 汇编指令对比

| line | 8.4.0 | 10.2.0 |
|------|----------------------|----------------------|
| 1 | movl 252(%rbp), %eax | movl 252(%rbp), %edx |
| 2 | movslq %edx, %rdx | cltq |
| 3 | salq \$4, %rdx | salq \$4, %rax |
| 4 | leaq 272(%rbp), %rbx | leaq 272(%rbp), %rbx |
| 5 | addq %rbx, %rdx | addq %rbx, %rax |
| 6 | subq \$160, %rdx | subq \$160, %rax |
| 7 | movl (%rdx), %edx | movl (%rax), %eax |
| 8 | subl %edx, %eax | subl %eax, %edx |

表 1 展示了由 ‘8.4.0’ 和 ‘10.2.0’ 版本的 GCC 编译器生成的汇编文件中的部分差异。从表中可以看出，主要差异集中在操作码和寄存器的使用上。因此，我们计划提取汇编文件中的操作码和寄存器的数量。此外，考虑到某些可能的关联特性，如操作码与寄存器、寄存器与寄存器之间的关系，我们还将提取 Bigram 的数量。其中 Bigram (2-gram) 是指连续两个词或字符的组合，例如 “I am a student” 的 bigram 为 [‘I am’, ‘am a’, ‘a student’]。

图 3 展示了由 ‘10.2.0’ 和 ‘12.2.0’ 版本的 GCC 编译器生成的汇编文件中的部分差异。从图中可以看出，主要差异体现在汇编代码块的位置调度上。为了捕捉这种位置特征，我们计划提取 block 的数量。block 定义为连续的汇编指令组合，经过实验验证，我们

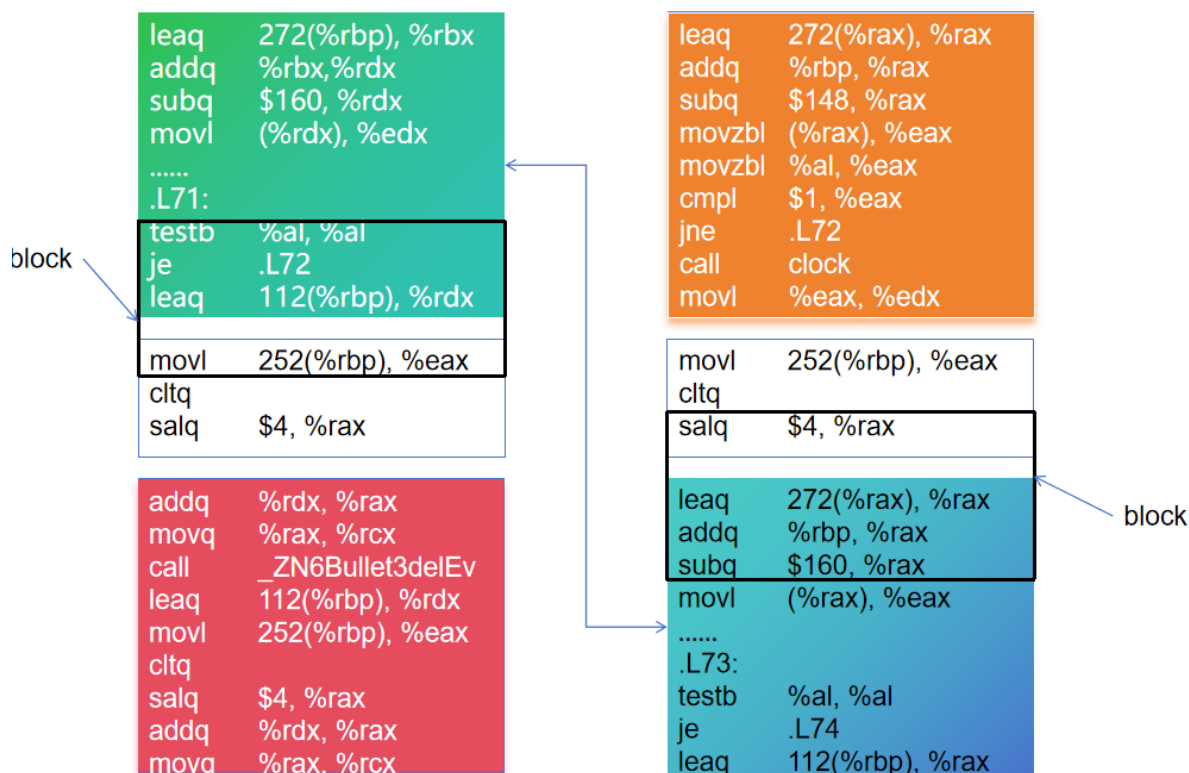


Figure 3: Enter Caption

选择连续 4 条汇编指令的组合效果最佳。

综上所述，我们将提取操作码、寄存器、Bigram 与 block 数量作为一个汇编文件的特征属性。

2.2 提取特征

对汇编文件进行特征提取之前，为了获取更加泛化有效的特征，我们对其做了一些预处理。

例如，在 8.4.0 版本 GCC 编译器生成的汇编文件中，指令'movl \$98, %eax'与'movl \$1, %eax'都表示将一个立即数移动到寄存器%eax 中，随后寄存器中的数值即为该立即数。因此，我们将汇编文件中的所有立即数统一替换为 \$0，替换后的汇编指令形如'movl \$0, %eax'。对于指令'movl -12(%rbp), %eax'与'movl -20(%rbp), %eax'，两者都表示将位于基址寄存器%rbp 偏移一定字节的内存地址中的 32 位数据读取到寄存器%eax 中。因此，我们统一将这些指令中的偏移量删除，替换后的汇编指令形如'movl (%rbp), %eax'。

通过上述方法对汇编文件进行预处理，我们在保持其基本语义完整的前提下，能够更加有效地提取出更多相关的特征。具体来说，这些预处理步骤不仅规范了汇编指令的表示形式，还减少了特征的多样性，使得后续的特征提取和分析更加精准和高效。这种预处理方式有助于在特征选择和模型训练过程中，突出重要的特征关联，同时避免因不必要的语法差异而带来的干扰，从而提升模型的整体表现和分析的准确性。

我们将对生成的五个汇编文件进行特征提取，最终获得的数据规模为 5*428，这意

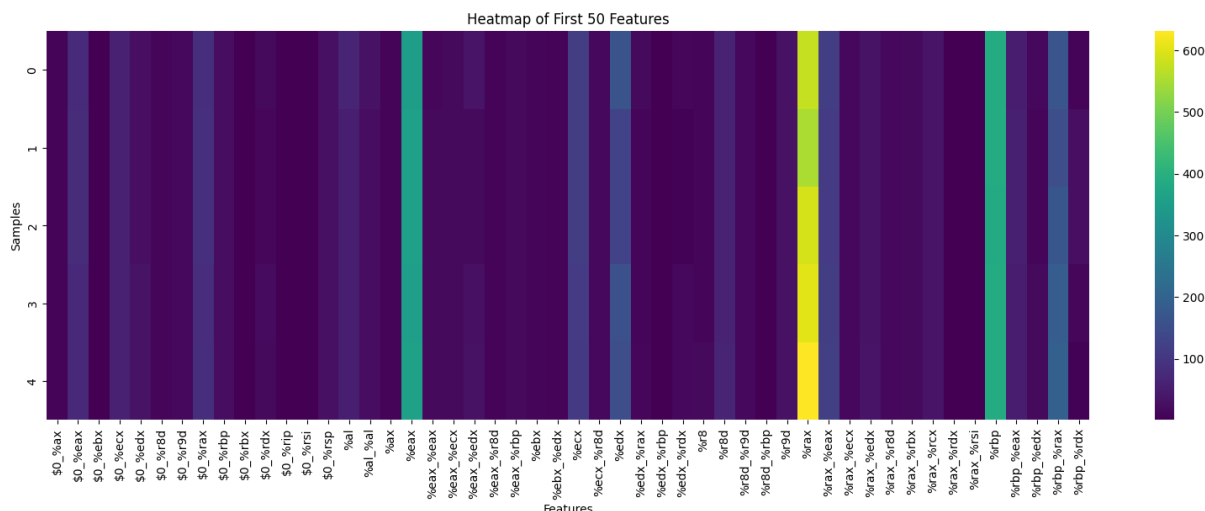


Figure 4: Enter Caption

意味着每个汇编文件包含 428 个特征。由于特征维度较大，为了便于分析和展示，我们将选择前 50 个特征数据，并使用热力图进行可视化。通过这种方式，我们可以更直观地观察和比较各汇编文件在这些特征维度上的差异和相似性，从而为后续的数据分析提供有价值的参考。

从图 4 可以看出，五个汇编文件在某些特征维度上的取值大致相同。鉴于此，我们决定对数据进行降维处理，将这些取值相同的特征去除。这样做有两个主要原因：一方面，减少特征维度可以降低模型的计算复杂度，简化模型的训练过程；另一方面，取值相同的特征对我们后续的版本判别并无实际帮助。因此，在降维之后，我们最终获得了规模为 5×89 的数据集，这意味着共有 339 个特征的取值是相同的，并已被剔除。结果如表 2 所示：

Table 2: 特征数据

| feature_name | 8.4.0 | 10.2.0 | 11.3.0 | 12.2.0 | 13.2.0 |
|---------------|-------|--------|--------|--------|--------|
| \$0_%eax | 77 | 78 | 78 | 72 | 72 |
| \$0_%edx | 25 | 25 | 25 | 31 | 31 |
| \$0_%rax | 84 | 89 | 89 | 80 | 84 |
| \$0_%rdx | 17 | 12 | 12 | 19 | 17 |
| %al2 | 64 | 55 | 55 | 55 | 55 |
| ... | ... | ... | ... | ... | ... |
| testb | 30 | 21 | 21 | 21 | 21 |
| testb_%al | 30 | 21 | 21 | 21 | 21 |
| testb_je_leaq | 9 | 8 | 8 | 8 | 8 |
| testl | 2 | 6 | 6 | 6 | 6 |
| testl_%eax | 2 | 6 | 6 | 6 | 6 |

3 问题二模型的建立与求解

3.1 数据分析

我们画出部分特征数据的分布，得到结果如图 5 所示。从图中我们可以看出，不同版本 GCC 编译器生成的汇编文件对于操作码、寄存器、Bigram 与 block 数量存在一定的差异，而这也正是我们去判断编译器版本的重要依据。

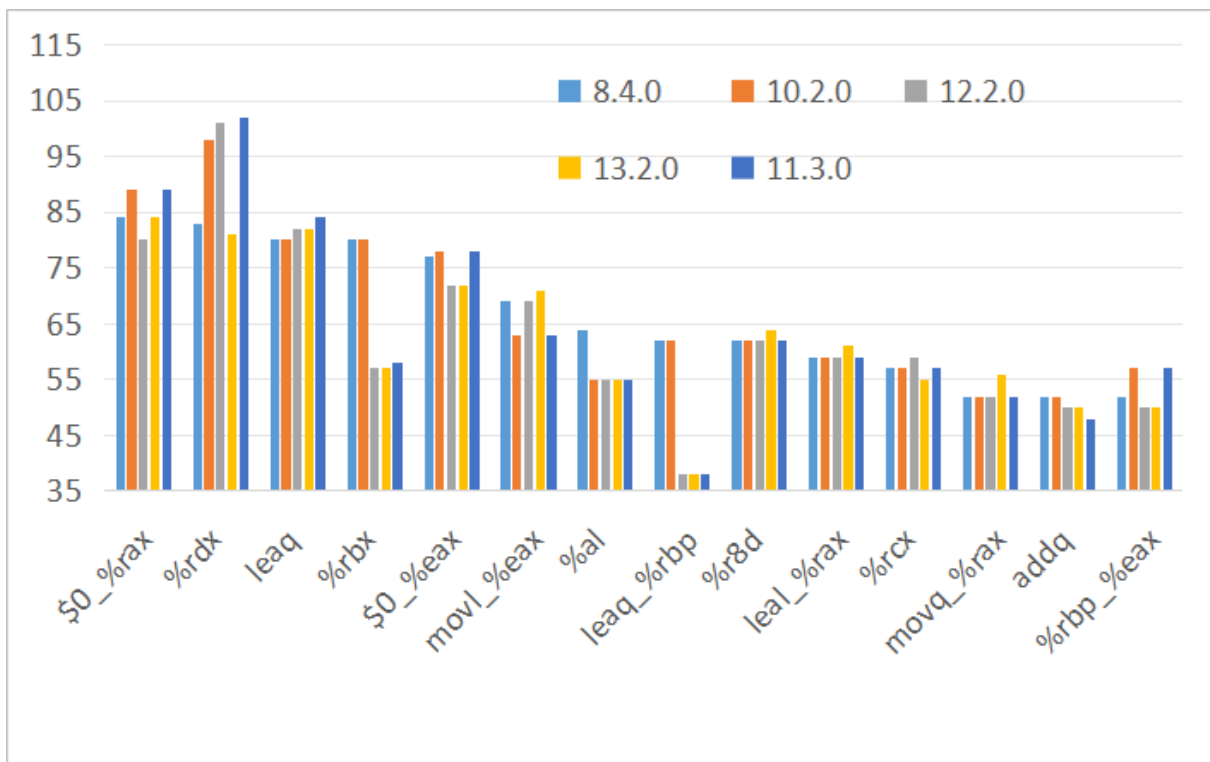


Figure 5: Enter Caption

计算各个特征与标签的皮尔逊相关系数（Pearson Correlation Coefficient），该系数可以用来衡量适用于衡量两个连续变量之间的线性关系，皮尔逊相关系数的计算公式为

$$\rho_{X,Y} = \frac{\text{cov}(X,Y)}{\sigma_X \sigma_Y} \quad (1)$$

其中，协方差 $\text{cov}(X,Y)$ 的计算公式为：

$$\text{cov}(X,Y) = \frac{1}{n} \sum_{i=1}^n (X_i - \mu_X)(Y_i - \mu_Y) \quad (2)$$

标准差 σ_X 和 σ_Y 的计算公式为：

$$\sigma_X = \sqrt{\frac{1}{n} \sum_{i=1}^n (X_i - \mu_X)^2} \quad (3)$$

$$\sigma_Y = \sqrt{\frac{1}{n} \sum_{i=1}^n (Y_i - \mu_Y)^2} \quad (4)$$

均值 μ_X 和 μ_Y 的计算公式为:

$$\mu_X = \frac{1}{n} \sum_{i=1}^n X_i \quad (5)$$

$$\mu_Y = \frac{1}{n} \sum_{i=1}^n Y_i \quad (6)$$

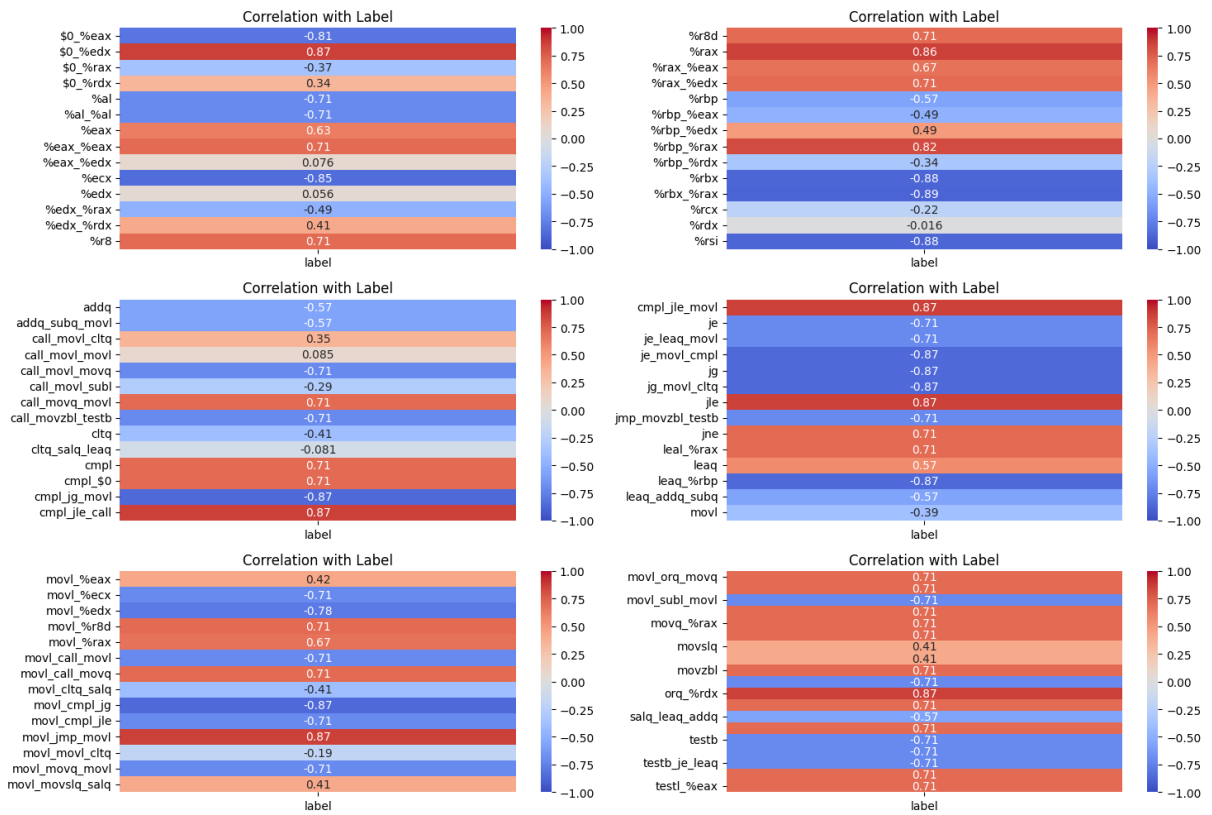


Figure 6: 特征与标签的皮尔逊相关系数

得到的结果如图 6 所示, 虽然图片中部分特征的相关系数较高, 但是我们进行了统计显著性检验。我们设原假设 H_0 是“两个变量之间不存在线性关系”, 备择假设 H_1 是“两个变量之间存在线性关系”。

我们使用 t 统计量来进行相关系数检验。 t 统计量的计算公式为:

$$t = \frac{r\sqrt{n-2}}{\sqrt{1-r^2}} \quad (7)$$

其中:

- r 是计算得到的 Pearson 相关系数。

- n 是样本量。
- t 统计量服从自由度为 $n - 2$ 的 t 分布。

计算得到 t 统计量后, 可以通过查找 t 分布表或使用计算工具来找到对应的 p 值。 p 值是与 t 统计量对应的累积概率值, 表示在原假设为真的情况下, t 统计量出现当前值或更极端值的概率。对于 p 值的解释是:

- **低 p 值 (通常 < 0.05):** 意味着在原假设为真的情况下, 观察到当前相关系数的概率非常低, 因此我们有理由拒绝原假设, 认为相关系数是显著的。
- **高 p 值 (通常 > 0.05):** 意味着在原假设为真的情况下, 观察到当前相关系数的概率较高, 因此我们无法拒绝原假设, 认为相关系数可能不是显著的。

我们可以从图 7 中看到几乎所有的特征属性的 p 值都大于显著性水平 $= 0.05$, 则不拒绝原假设, 认为相关系数可能是由于随机误差引起的, 不具有统计显著性。

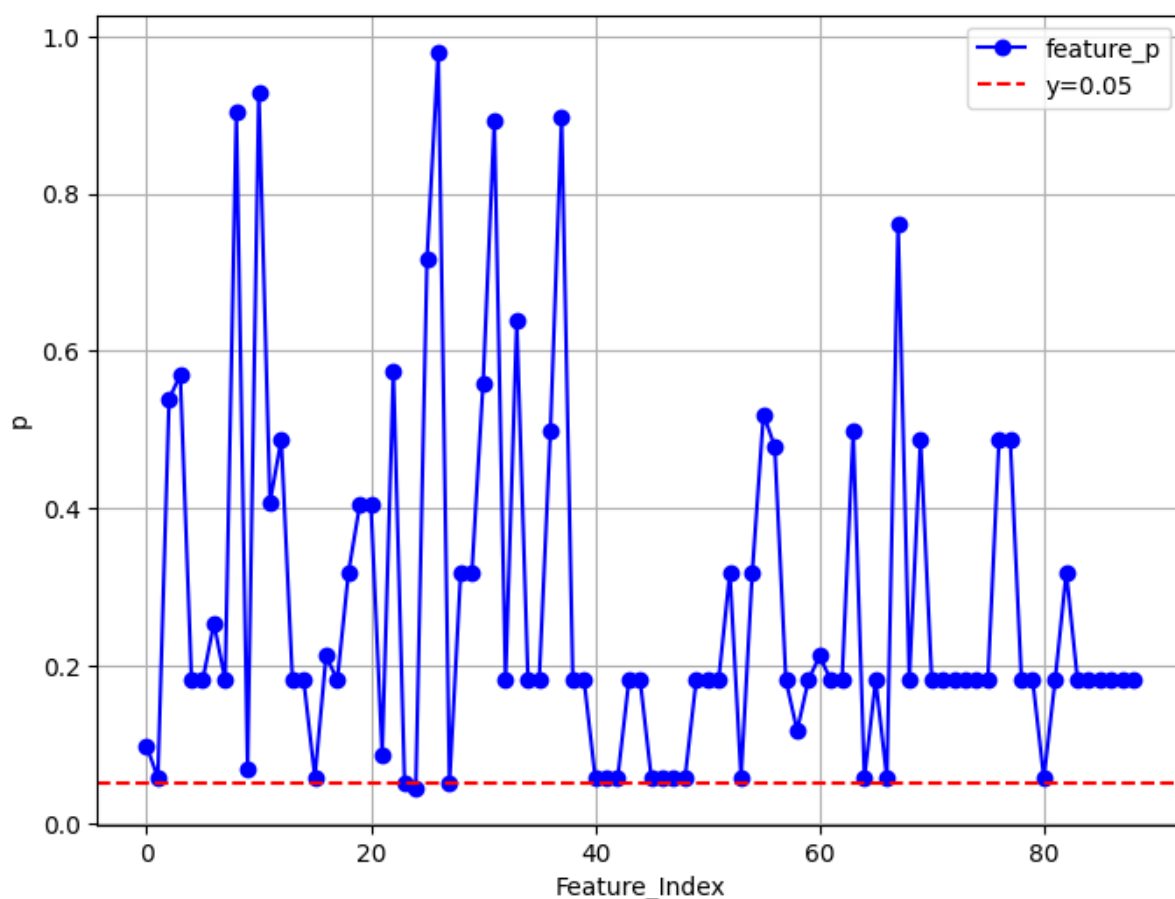


Figure 7: 统计显著性水平

因此我们接着计算了特征属性与标签的互信息。互信息衡量了两个随机变量之间的“信息共享”量。如果两个随机变量 X 和 Y 是独立的, 那么它们之间的互信息为零, 表

示知道一个变量不会提供关于另一个变量的任何信息。如果互信息较大，则表示两个变量之间有较强的依赖关系。

对于两个离散随机变量 X 和 Y ，互信息定义为：

$$I(X; Y) = \sum_{y \in Y} \sum_{x \in X} p(x, y) \log \left(\frac{p(x, y)}{p(x)p(y)} \right) \quad (8)$$

互信息还可以通过熵来表示：

$$I(X; Y) = H(X) + H(Y) - H(X, Y) \quad (9)$$

其中，各个熵的计算公式为：

$$H(X) = - \sum_{x \in X} p(x) \log p(x) \quad (10)$$

$$H(Y) = - \sum_{y \in Y} p(y) \log p(y) \quad (11)$$

$$H(X, Y) = - \sum_{x \in X} \sum_{y \in Y} p(x, y) \log p(x, y) \quad (12)$$

从图 8 中我们可以得到如下信息：

- 高互信息的特征：互信息值高的特征（如 `'%edx_%eax'`、`'%r8d'`、`'cmp_jle_movl'` 等）在图中以黄色或绿色显示，这些特征是我们模型中特别需要关注的，因为它们对标签具有较高的预测能力。
- 低互信息或零互信息的特征：互信息值为零或接近零的特征对标签几乎没有贡献，可以考虑在特征选择过程中进行剔除或降低其权重，以简化模型。
- 互信息的分布：从图中可以看出，互信息较高的特征集中在一些特定的寄存器组合或指令组合上，这表明这些组合对于模型预测特别重要。基于此，我们可以重点关注这些特征，并在模型训练中对其进行优化。

因此，我们根据数据分析以及阅读的论文，猜想特征属性与编译器版本之间存在的关系应该是非线性关系。而由于我们的数据是连续值，且考虑到不同源代码生成的汇编文件可能存在不同的特征属性，所以我们的模型还要求支持对含有缺失值的数据进行预测，综合考量，我们决定使用 Gini 决策树，该决策树可以很好的支持上述特性。

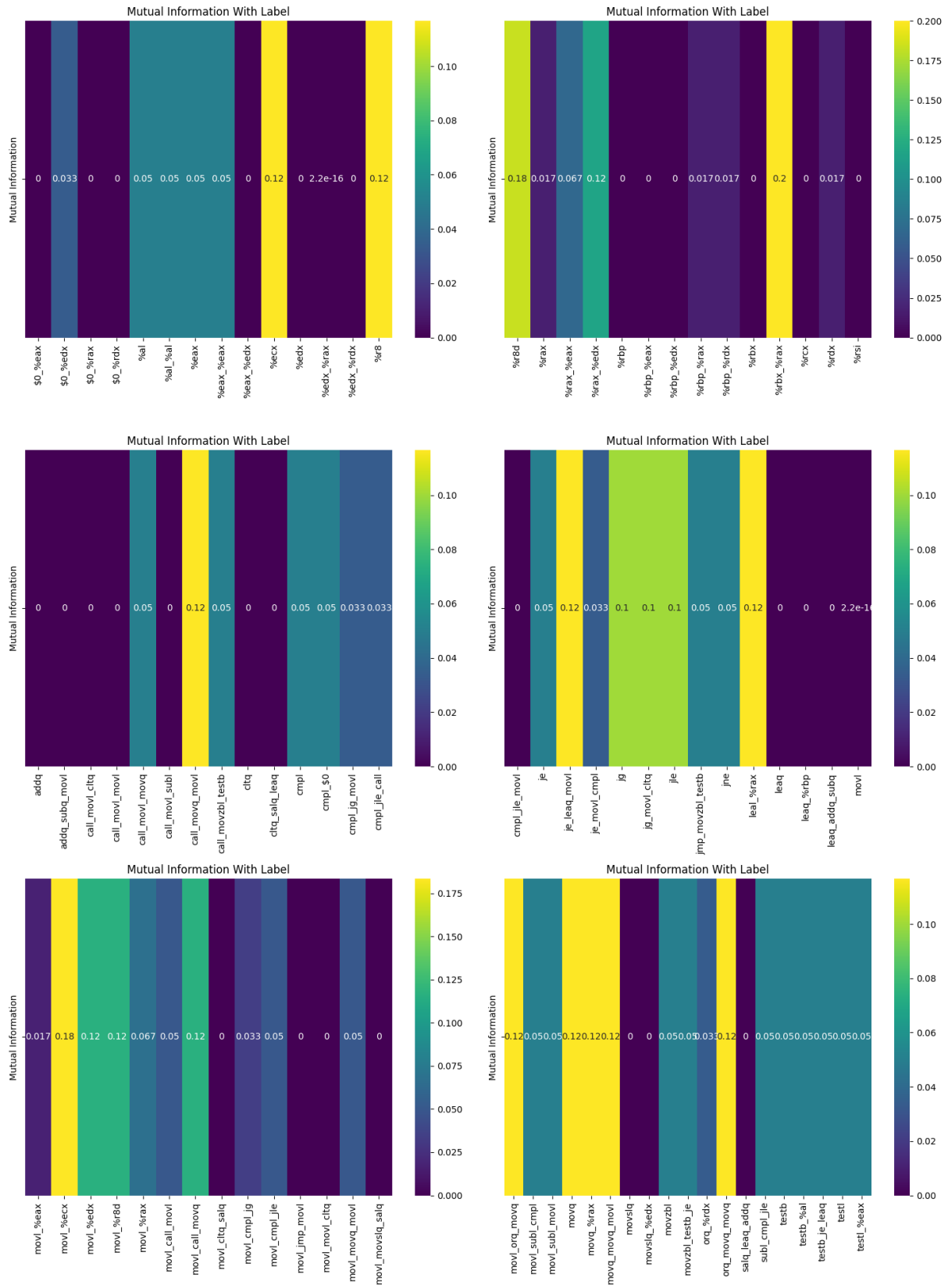


Figure 8: 特征与标签的互信息

3.2 模型说明

3.2.1 Gini 系数的计算

Gini 系数是衡量数据集中样本纯度的指标。对于给定的样本集合 D , Gini 系数可以表示为:

$$Gini(D) = 1 - \sum_{i=1}^C p_i^2 \quad (13)$$

其中:

- p_i 是样本集中第 i 类的样本比例 (即属于第 i 类的样本数量占总样本数量的比例)。
- C 是类别的数量。

3.2.2 连续属性划分点 t 的确定

排序连续属性的值于一个连续属性 A , 首先对所有样本的该属性值进行排序。假设排序后的值为 a_1, a_2, \dots, a_n , 其中 a_i 是第 i 个样本在属性 A 上的取值, 并且满足 $a_1 \leq a_2 \leq \dots \leq a_n$ 。

候选划分点的确定 候选划分点 t_i 通常选在两个相邻样本值的中间, 即:

$$t_i = \frac{a_i + a_{i+1}}{2}, \quad i = 1, 2, \dots, n-1 \quad (14)$$

计算每个划分点的 Gini 指数 对于每个候选划分点 t_i , 将数据集 D 分为两个子集 D_{left} 和 D_{right} :

$$D_{\text{left}} = \{x \in D \mid A(x) \leq t_i\} \quad (15)$$

$$D_{\text{right}} = \{x \in D \mid A(x) > t_i\} \quad (16)$$

然后计算基于划分点 t_i 的 Gini 指数:

$$Gini(D, A, t_i) = \frac{|D_{\text{left}}|}{|D|} Gini(D_{\text{left}}) + \frac{|D_{\text{right}}|}{|D|} Gini(D_{\text{right}}) \quad (17)$$

选择最优划分点 最优划分点 t^* 是使得 Gini 指数最小的划分点:

$$t^* = \arg \min_{t_i} Gini(D, A, t_i) \quad (18)$$

通过此方法, 我们得到了最优的划分点 t^* , 并将其用于决策树的构建。

3.2.3 处理连续属性

对于连续属性 A ，通过划分点 t 将样本集 D 划分为两个子集： D_{left} 和 D_{right} ，分别表示小于等于 t 和大于 t 的样本集。划分后的 Gini 指数为：

$$Gini(D, A, t) = \frac{|D_{\text{left}}|}{|D|} Gini(D_{\text{left}}) + \frac{|D_{\text{right}}|}{|D|} Gini(D_{\text{right}}) \quad (19)$$

其中：

- $|D_{\text{left}}|$ 和 $|D_{\text{right}}|$ 是两个子集的样本数量。
- $Gini(D_{\text{left}})$ 和 $Gini(D_{\text{right}})$ 分别是两个子集的 Gini 系数。
- $|D|$ 是样本集 D 的总样本数量。

3.2.4 处理缺失值

对于属性 A 存在缺失值的情况，假设缺失值比例为 p_{missing} ，则 Gini 系数可以表示为：

$$Gini(D, A) = (1 - p_{\text{missing}}) \times Gini(D_{\text{not-missing}}, A) + p_{\text{missing}} \times Gini(D_{\text{missing}}) \quad (20)$$

其中：

- $D_{\text{not-missing}}$ 是属性 A 有值的样本集。
- D_{missing} 是属性 A 缺失值的样本集。
- $Gini(D_{\text{not-missing}}, A)$ 和 $Gini(D_{\text{missing}})$ 分别是有值样本集和缺失值样本集的 Gini 系数。

3.2.5 选择最优划分

在考虑了所有可能的划分点 t 以及缺失值的影响后，选择使得 Gini 指数最小的划分点作为最终的划分点，以确保在当前节点上的样本被分成更加纯净的子集。

3.2.6 构建树结构

在找到最优划分后，决策树会在此处创建一个分裂节点，并将数据集分成左右子集。然后递归地对每个子集重复上述步骤，直到满足停止条件为止。

3.3 模型建立

Gini 决策树的伪代码如下:

Algorithm 1 决策树构建算法

Require: 数据集 D , 属性集 A , 停止条件

Ensure: 决策树

函数 $\text{BuildTree}(D, A)$:

If 所有样本都属于同一类 **Then**

 将当前节点标记为该类, 返回

If 属性集为空或满足停止条件 **Then**

 将当前节点标记为样本中最多的类, 返回

For each 属性 $A_i \in A$ **Do**

If A_i 是连续属性 **Then**

 对 A_i 的值进行排序

 计算每个候选划分点 $t_i = \frac{a_i + a_{i+1}}{2}$

 对每个候选划分点 t_i 计算 Gini 指数

Else If A_i 有缺失值 **Then**

 计算 Gini 指数, 考虑缺失值比例 p_{missing}

Else

 计算每个可能划分的 Gini 指数

 选择使 Gini 指数最小的属性 A^* 及其最优划分点 t^*

 将当前节点标记为 A^* 的分裂节点

 根据划分点 t^* 将样本集 D 分为 D_{left} 和 D_{right}

 递归调用 $\text{BuildTree}(D_{\text{left}}, A)$ 和 $\text{BuildTree}(D_{\text{right}}, A)$

End Function

基于伪代码[算法 1](#), 我们构建了一颗 Gini 决策树, 如图 9 所示:

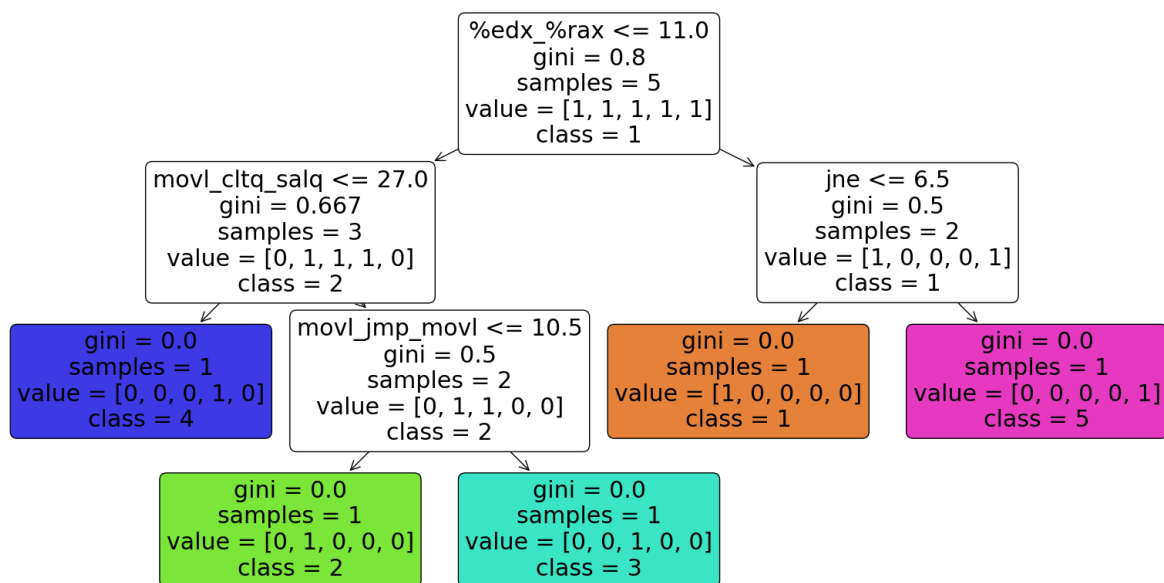


Figure 9: Gini 决策树

4 问题 3

4.1 决策树模型分析

决策树模型作为一种简单且直观的机器学习算法，广泛应用于分类和回归任务。然而，尽管其易于理解和解释，决策树模型也存在一些固有的缺陷，决策树模型存在以下几个缺陷：

- 容易过拟合：决策树模型在处理复杂数据集时，容易出现过拟合问题。这是因为决策树模型在构建过程中会根据数据的分布来选择分裂点，如果数据集过于复杂，决策树可能会生成**过深的树结构**，从而导致模型过拟合，泛化能力较差。

图 10展示了不同拟合状态的大致情况，过拟合是模型完全拟合了训练数据，把训练的数据的噪声也当做数据的统计特征，从而导致在训练集的误差很低，但是在测试集的误差很高。

- 对数据噪声敏感：决策树对数据中的噪声和异常值非常敏感。这是因为决策树在构建过程中会根据数据的分布来选择分裂点，噪声和异常值可能导致树的结构发生较大变化，从而影响模型的稳定性和准确性。
- 不稳定：由于决策树的构建过程是基于局部最优的贪心算法，因此对数据的变化非常敏感，小的输入数据变化可能导致决策树结构的显著变化。
- 计算复杂度：在特征数量较多或数据集较大时，决策树的构建过程可能会非常耗时。这是因为每次分裂都需要计算所有可能分裂点的增益，从而选择最佳分裂点。



Figure 10: 不同拟合状态

并且，在我们的实验中，我们发现如果只使用附件 1 或者是附件 2 来构建模型，模型的预测结果将会趋向于随机猜测，而非根据编译器的特征进行预测。

从数据的分布来看, 我们使用 t-SNE 算法对数据进行降维可视化,t-分布邻域嵌入 (t-distributed Stochastic Neighbor Embedding, t-SNE) [1] 是一种非线性降维技术, 特别适用于高维数据的可视化。t-SNE 通过将高维数据映射到低维空间, 同时**尽可能保持数据点之间的局部结构**, 从而揭示数据的内在结构和模式。t-ESN 算法通过优化 KL 散度来最小化高维空间和低维空间之间的分布差异, 从而实现降维。t-SNE 的优化目标是**最小化高维空间和低维空间之间的 KL 散度**, KL 散度定义如下:

$$C = KL(P||Q) = \sum_i \sum_j p_{ij} \log \frac{p_{ij}}{q_{ij}}, \quad (21)$$

其中, p_{ij} 是高维数据点 x_i 和 x_j 之间的相似度, 在高维空间中, t-SNE 算法使用高斯分布来衡量高维空间中点对之间的相似性。 q_{ij} 是低维数据点 y_i 和 y_j 之间的相似度。在低维空间中, t-SNE 使用学生 t 分布 (t-distribution) 来计算点对之间的相似性。这个分布的长尾特性使得它能够更好地捕捉低维空间中的局部结构。t-SNE 通过梯度下降算法优化该目标, C 对于低维数据的梯度为

$$\frac{\delta C}{\delta y_i} = 4 \sum_j (p_{ij} - q_{ij})(y_i - y_j) (1 + \|y_i - y_j\|^2)^{-1}, \quad (22)$$

最后使用梯度更新低维数据点 y_i

$$y_i^{(t+1)} = y_i^{(t)} + \eta \frac{\delta C}{\delta y_i}, \quad (23)$$

其中 η 是学习率。在 t-SNE 算法中, 还引入了**动量**来进一步优化低维数据点的表示:

$$momentum = \alpha(y_i^{(t)} - y_i^{(t-1)}). \quad (24)$$

动量项通过在更新过程中加入前几次更新的累积效果, 可以帮助算法在梯度方向上更快地移动, 并且通过平滑更新路径, 能够有效减少震荡, 使得优化过程更加稳定。加入动量后的更新公式为

$$y_i^{(t+1)} = y_i^{(t)} + \eta \frac{\delta C}{\delta y_i} + \alpha(y_i^{(t)} - y_i^{(t-1)})m \quad (25)$$

综上所述, T-SNE 算法流程如[算法 2](#)所示。

Algorithm 2 t-SNE**Require:** 数据集 X , 目标维度 k , 优化参数: 迭代次数 T , 学习率 η , 动量参数 α **Ensure:** 低维数据 \mathcal{Y} 计算高维数据之间的相似度 $p_{j|i}$

$$p_{j|i} = \frac{\exp(-\|x_i - x_j\|^2 / 2\sigma_i^2)}{\sum_{k \neq i} \exp(-\|x_i - x_k\|^2 / 2\sigma_i^2)}$$

其中 σ_i 是以 \mathbf{x}_i 为中心的高斯分布的方差, 通过二分搜索确定
 从 $\mathcal{N}(0, 10^{-4}I)$ 从采样 $\mathcal{Y}^{(0)} = y_1, y_2, \dots, y_n$

for $i = 1$ to T **do**计算低维数据之间的相似度 $q_{j|i}$

$$q_{j|i} = \frac{(1 + \|y_i - y_j\|^2)^{-1}}{\sum_{k \neq i} (1 + \|y_i - y_k\|^2)^{-1}}$$

计算梯度

$$\frac{\delta C}{\delta y_i} = 4 \sum_j (p_{ij} - q_{ij})(y_i - y_j) (1 + \|y_i - y_j\|^2)^{-1}.$$

更新 y_i

$$y_i^{(t+1)} = y_i^{(t)} + \eta \frac{\delta C}{\delta y_i} + \alpha(y_i^{(t)} - y_i^{(t-1)})$$

end for

我们使用 Scikit-learn[2] 实现 t-SNE 算法的可视化, 结果如图 11 所示。

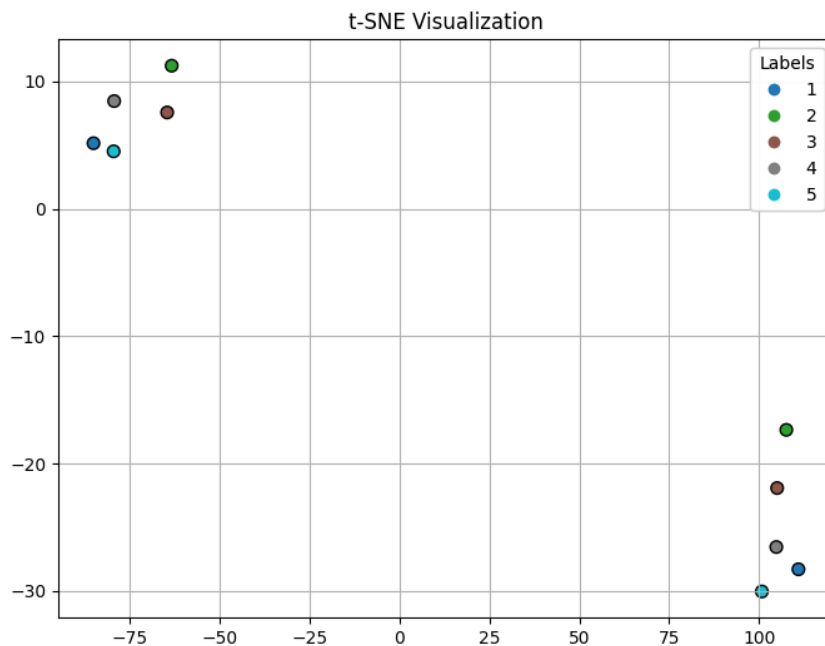


Figure 11: t-SNE 降维可视化

可以发现, 附件 1 和附件 2 在整体上的数据分布相差很远, 并且在局部上同一个源

文件的不同版本之间也没有明显的分布规律。这也是导致决策树模型无法准确预测的原因之一。

4.2 模型改进

通过对决策树模型和数据分布的分析,我们发现以下问题

- **训练数据不具备广泛性:** 原有的训练数据少,并且类型单一,无法体现编译器在编译技术上的优化方法和特点
- **模型本身的缺陷:** 单纯的决策树模型容易过拟合: 在没有剪枝方法的情况下,决策树的深度容易过深,导致模型过拟合,泛化能力较差。但是如果使用剪枝方法,则决策树又容易欠拟合,导致模型的准确率较低

针对训练的问题,我们需要根据不同版本编译器的特点来对应生成数据。使用对应生成的数据训练模型,可以使得模型能够捕捉到不同版本编译器在编译技术和优化方法上的差异。我们注意到一个软件:**CSmith**。该软件是用于编译器压力测试的随机程序生成器。它专门用于生成程序源代码,以帮助测试和验证编译器的正确性和稳健性。生成的代码有以下特点:

- 1.CSmith 生成的代码涵盖了编程语言的大部分特性,包括指针、数组、结构体以及复杂的控制流结构等。这种广泛的覆盖性有助于发现不同版本编译器在编译技术上的差异。
- 2. 随机性和多样性: 生成的代码是随机的,这意味着每次运行 CSmith 都会产生不同的程序。这进一步提高生成的代码的随机性和多样性。

基于 CSmith,我们生成了 1000 个随机程序,并使用不同版本的编译器对这些程序进行编译,得到了 1000 个数据样本。

由于我们引入了大量数据,并且决策树模型存在固有的缺陷,因此我们考虑使用**集成模型**来构建判别函数。集成模型是一种强大的机器学习方法,首先构建多个弱分类器,然后在把这些弱分类器的分类结果进行加权,得到一个强分类器。[图 12](#)显示了集成模型的基本原理。在选择机器学习模型时,我们主要考虑以下几个关键特性:**训练速度快、能够处理高维数据以及具备良好的泛化能力**。基于这些特性,我们选择了 **XGBoost** 模型。**XGBoost (Extreme Gradient Boosting)** [\[3\]](#) 是一种高效且灵活的梯度提升框架,通过集成决策树模型并逐步优化,展现出卓越的性能和可扩展性,尤其在处理大规模数据集时表现优异。XGBoost 的主要优点包括:

- **高性能:** XGBoost 通过直方图优化、GPU 加速以及数据/特征并行等技术,大幅提升了训练效率,相较于一般的集成模型,它能够更快速地完成训练过程。

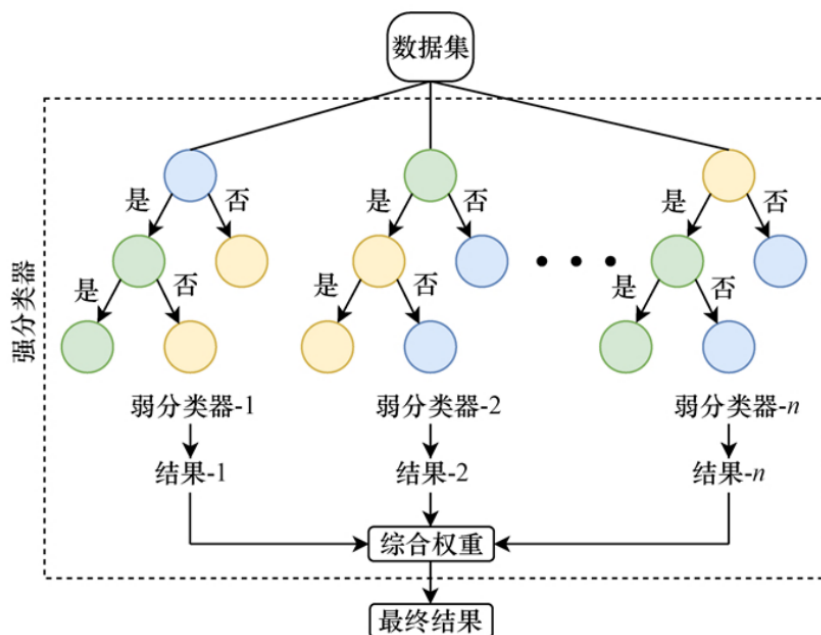


Figure 12: 集成模型的基本原理

- 灵活性：作为一个集成决策树模型，XGBoost 不仅能够自动处理缺失值，还能处理数值型和类别型等多种类型的数据。这种灵活性使其非常契合我们数据集的特点。
- 准确性：XGBoost 通过拟合决策树的预测残差，逐步优化树模型的结构和权重，从而提升预测准确性。这种逐步优化的方式有效地提高了模型的整体性能。
- 可解释性：XGBoost 提供了特征重要性评估和模型解释功能，使我们能够更直观地理解模型的预测结果，以及各特征对预测结果的影响。这种可解释性对于模型的应用和改进具有重要意义。

通过这些优势，XGBoost 成为我们在处理复杂数据集和追求高效、准确预测时的理想选择。XGBoost 的原理如下：假设已经训练了 $t-1$ 个模型，现在准备增加一个模型 $f_t(x)$ ，使得目标函数：

$$\mathcal{L}(F_{t-1} + \alpha f_t) = \sum_{i=1}^n l(y_i, \hat{y}^{(t)}) + \Omega(f_t), \quad (26)$$

最小化，其中 $\hat{y}^{(t)} = F_{t-1}(x) + f_t(x)$ ， $\Omega(f_t)$ 是正则化项， $l(y_i, \hat{y}^{(t)})$ 是损失函数， n 是样本数量。我们对损失函数 $l(y_i, \hat{y}^{(t)})$ 在 $F_{t-1}(x) + f_t(x)$ 处进行二阶泰勒展开，并且忽略余项，那么我们可以得到

$$l(y_i, \hat{y}^{(t)}) \approx \sum_{i=1}^n \left[l(y_i, \hat{y}_i^{(t-1)}) + g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i) \right], \quad (27)$$

其中 $g_i = \nabla_{\hat{y}^{(t-1)}} l(y_i, \hat{y}_i^{(t-1)})$ $h_i = \nabla_{\hat{y}^{(t-1)}}^2 l(y_i, \hat{y}_i^{(t-1)})$, 对于每个叶子结点的预测

$$f_t(x) = w_{q(x)}, w \in \mathbb{R}^T, q: \mathbb{R}^d \mapsto \{1, 2, \dots, T\}, \dots, \quad (28)$$

其中 T 是叶子结点的个数, $q(x)$ 是样本 x 在树中的叶子结点的索引 (因为每个样本最终都会落到某个叶子节点上), w 是叶子结点的权重, 那么我们就可以定义正则化函数:

$$\Omega(f_t) = \gamma T + \frac{1}{2} \lambda \sum_{i=1}^T w_i^2, \quad (29)$$

γ 和 λ 是超参数, 分别控制叶子结点的个数和叶子结点的惩罚权重, 最终的优化目标为

$$\begin{aligned} \mathcal{L}^{(t)} &\simeq \sum_{i=1}^n \left[l(y_i, \hat{y}_i^{(t-1)}) + g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i) \right] + \Omega(f_t) \\ &= \sum_{i=1}^n \left[g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i) \right] + \gamma T + \frac{1}{2} \lambda \sum_{j=1}^T w_j^2 + co \\ &= \sum_{j=1}^T \left[\left(\sum_{i \in I_j} g_i \right) w_j + \frac{1}{2} \left(\sum_{i \in I_j} h_i + \lambda \right) w_j^2 \right] + \gamma T + const \end{aligned} \quad (30)$$

其中 I_j 是样本集合 $\{i | q(x_i) = j\}$, 也就是由于每个样本最终都会落到某个叶子节点上, 因此把对样本的求和转换为对叶子结点的求和, 并且 $l(y_i, \hat{y}_i^{(t-1)})$ 是已经确定的常数, 因此最终优化目标是

$$\mathcal{L}^{(t)} = \sum_{j=1}^T \left[\left(\sum_{i \in I_j} g_i \right) w_j + \frac{1}{2} \left(\sum_{i \in I_j} h_i + \lambda \right) w_j^2 \right] + \gamma T \quad (31)$$

这是一个关于 w_i 的二次函数, 可以通过求导得到最优解, 令

$$G_j = \sum_{i \in I_j} g_i \quad H_j = \sum_{i \in I_j} h_i, \quad (32)$$

则优化目标的闭式解为

$$w_j^* = -\frac{G_j}{H_j + \lambda} \quad \mathcal{L}^{(t)} = -\frac{1}{2} \sum_{j=1}^T \frac{G_j^2}{H_j + \lambda} + \gamma T, \quad (33)$$

那么在构建决策树时, 我们可以通过贪心算法, 从根节点开始, 对每个叶子结点, 计算其最优的权重 w_j , 然后计算损失函数,

$$\text{Gain} = \frac{1}{2} \left(\frac{G_L^2}{H_L + \lambda} + \frac{G_R^2}{H_R + \lambda} - \frac{(G_L + G_R)^2}{H_L + H_R + \lambda} \right) - \gamma, \quad (34)$$

其中 G_L, H_L 和 G_R, H_R 分别是左右子树的梯度值, 最后选择最优的划分特征和划分点, 递归构建树, 直到满足停止条件。在本题中, 由于是分类问题, 我们使用交叉熵作为损失函数,

$$l(y_i, \hat{y}_i^{(t-1)}) = - \left(y_i \log(\hat{y}_i^{(t-1)}) + (1 - y_i) \log(1 - \hat{y}_i^{(t-1)}) \right) \quad (35)$$

则该损失函数的梯度和二阶导数分别为

$$g_i = \hat{y}_i^{(t-1)} - y_i, \quad h_i = \hat{y}_i^{(t-1)}(1 - \hat{y}_i^{(t-1)}) \quad (36)$$

最终, XGBoost 的算法如[算法 3](#)所示。

Algorithm 3 XGBoost

Require: 超参数: λ 和 γ 。特征 X , 标签 Y 。停止条件: 树的最大深度 K 。

Ensure: 集成模型 XGBoost

- 1: **计算梯度和二阶导数:**
 - 2: **for** 每个样本 i **do**
 - 3: 计算损失函数的一阶梯度 g_i 和二阶导数 h_i 。
 - 4: **end for**
 - 5: **遍历所有特征:**
 - 6: **for** 每个特征 j **do**
 - 7: 计算所有可能的分裂点。
 - 8: **for** 每个分裂点 **do**
 - 9: 将数据集分为左子集 L 和右子集 R 。
 - 10: 计算左子集和右子集的梯度和二阶导数之和:
 - 11: $G_L = \sum_{i \in L} g_i, \quad H_L = \sum_{i \in L} h_i$
 - 12: $G_R = \sum_{i \in R} g_i, \quad H_R = \sum_{i \in R} h_i$
 - 13: 计算分裂增益:
 - 14: $\text{Gain} = \frac{1}{2} \left(\frac{G_L^2}{H_L + \lambda} + \frac{G_R^2}{H_R + \lambda} - \frac{(G_L + G_R)^2}{H_L + H_R + \lambda} \right) - \gamma$
 - 15: **end for**
 - 16: **end for**
 - 17: **选择最佳分裂点:**
 - 18: 对于每个特征, 选择具有最大增益的分裂点。
 - 19: 在所有特征中, 选择增益最大的分裂点作为最终的分裂点。
 - 20: **递归分裂:**
 - 21: 对于新的左子树和右子树, 重复步骤 2 至 5, 直到满足停止条件
-

对于 d 个特征, 每个特征, 都可以以 $O(1)$ 的速度构建计算增益, 每一层都需要 $n \log n$ 的时间排序选出最佳特征, 一共 K 层, 因此整个 XGBoost 算法的时间复杂度为 $O(Kdn \log n)$

5 问题四

提高由编译结果区分编译器版本的判别函数性能的建议:

5.1 自动特征提取

特征工程将原始数据转化为更适合模型学习的特征的过程, 直接影响模型的性能和准确性。然而, 传统的手动特征工程存在一些缺陷。首先, 手动特征工程依赖于领域专家的知识 and 经验, 这将导致特征选择的主观性和局限性。其次, 手动提取特征通常是一个耗时且繁琐的过程, 尤其在 高维数据或复杂数据结构的情况下, 难以保证全面和高效。因此, 如果 要提高模型的区分度和性能, 我们可以考虑采用深度学习技术来进行自动特征提取。我们构想了一个基于协同注意力机制的模型。具体来说, 首先将源文件和对应的汇编文件转换为词向量嵌入表示。然后, 利用基于 Transformer 的自注意力机制来捕捉文件内部的语义特征。为了更好地理解源文件与汇编文件之间的关系, 进一步采用 Cross-Attention 机制来提取它们之间的语义相似性和结构相异性。这种方法的优势在于, 它能够自动捕捉不同编译器版本之间在语义和结构上的细微差异, 这些差异可以作为有效的特征, 用于构建高效的分类模型。

5.2 增加数据集多样性

考虑到不同领域的源代码会存在很大的差异, 例如同为 C++ 程序代码, 后端服务器代码往往倾向于网络和 IO 操作, 而游戏的代码则会包含大量的图形渲染和物理计算, 因此不同领域的源代码在同一个版本的编译器编译的编译结果存在很大差异。此外, 即使是同一个领域的源代码, 不同规模的源代码之间也会存在非编译器造成的差异。最后编译器版本之间的差距主要是体现在对于不同版本的语言支持以及静态代码优化技术, 而后者一般要通过编译指令开启代码优化才能体现。鉴于以上分析, 我们可以通过以下几种方式来增加数据集的多样性, 提高模型的泛化能力:

- 不同类型的程序: 收集多种类型的程序 (如计算密集型、IO 密集型、混合型) 进行编译, 提取汇编结果。
- 不同规模的程序: 从小规模到大规模的程序进行编译, 确保模型能泛化到不同复杂度的代码。
- 编译优化级别: 使用不同的优化级别 (如-O0, -O1, -O2, -O3) 进行编译, 捕捉编译器在不同优化下的行为差异。

5.3 结合静态和动态特征

除了源代码和汇编代码之外, 我们还可以考虑结合静态特征和动态特征来提高模型的区分度。具体来说, 我们可以提取以下两类特征:

- 静态特征：提取静态汇编特征，如指令频率、寄存器使用率、基本块大小。
- 动态特征：运行程序，收集运行时的行为特征，如执行路径、内存访问模式、CPU 使用率。我们可以通过结合静态和动态的特征进行分析，更全面地了解编译器版本之间的差异，提高模型的泛化能力。

References

- [1] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9(86):2579–2605, 2008.
- [2] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.
- [3] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, page 785–794, New York, NY, USA, 2016. Association for Computing Machinery.
- [4] M. White. Isaac newton: the last sorcerer. *Macmillan*.
- [5] Rob Iliffe and George E. Smith. The cambridge companion to newton || a brief introduction to the mathematical work of isaac newton. 10.1017/CCO9781139058568(9):382–420, 2016.
- [6] Hita Kambhamettu, John Billos, Tomi Oluwaseun-Apo, Benjamin Gafford, Rohan Padhye, and Vincent J. Hellendoorn. On the naturalness of fuzzer-generated code. *2022 IEEE/ACM 19th International Conference on Mining Software Repositories (MSR)*, pages 506–510, 2022.