

软件需求工程 实验二：需求排序

- 一、小组成员及得分分配
- 二、实验目的
- 三、实验数据
- 四、实验方法
  - 1. 需求排序方法
  - 2. 需求排序性能评价指标
- 五、实验结果
  - 1. 需求排序结果
  - 2. 需求排序性能评价结果
- 六、结论
  - 1. 总体评价
  - 2. 误差分析
  - 3. 实验反思
- 七、参考文献

# 软件需求工程 实验二：需求排序

## 一、小组成员及得分分配

学号	姓名	成绩分配比例
181860031	胡欣然	0.33
181860058	刘雅婷	0.33
181860138	张问津	0.34

## 二、实验目的

我们选择具有丰富项目文档且具有GitHub Issue Tracker的知名开源项目：VSCode为分析对象，并在GitHub中VSCode项目下的Issue中进行数据获取。通过将排序问题转化为分类问题（P1~P4共四个等级），对所获取的需求进行排序。最后利用Kappa系数、海明距离等多种评价指标对排序结果进行评价与分析。








## 三、实验数据

实验一中，我们利用python直接对[github中VSCode的issues部分](#)进行了需求的爬取，获取到了open状态下前20页的issues和closed状态下前20页的issues，并在此基础上进行需求分析与分类。

然而，实验一中获取的需求没有关于需求排序的标签。如果我们在实验二中继续使用上述需求，则无法对排序结果的有效性进行性能评估。

幸运的是，我们在GitHub中VSCode的 Pinned issues 部分找到了[Iteration Plan for November 2020](#)。其中记录了维护团队在2020年11月的工作，并列举了一系列需求。这些需求被分为多个类别，且这些类别之间有较为明显的等级关系（如下图所示）。此外，已经完成的需求前被打上了“✓”。

链接中给定的标签

Mark	Description
	work in progress
	blocked task
	stretch goal for this iteration
	missing issue reference
	more investigation required to remove uncertainty
	under discussion within the team
	a large work item, larger than one iteration

在该链接列举的需求中，除了 finished，work in progress，blocked task 和 stretch goal for this iteration 四个状态外，剩余四个标签（即：上图中最后四行）几乎没有用到。于是，我们将需求分为P1到P4四个等级，分别对应：finished,work in progress,stretch goal for this iteration 和 blocked task 四个标签。最后将该链接中的需求信息进行了整理（包括：issue\_number,issues\_title,issues\_body,labels,issues\_status和level），并记录在 total.xlsx 中。

## 四、实验方法

### 1. 需求排序方法

首先，从功能的重要性、涉及到的用户体量、功能的使用频率和需求的可实现性这四个方面出发，制定了如下四条需求的评价标准：

需求优先级评价角度	需求优先级评价的定性原则
功能的重要性	主要功能优于次要功能
涉及到的用户体量	涉及较多用户的需求优于少量用户的需求
功能的使用频率	常用功能的需求优于不常用功能的需求
需求的可实现性	易实现的需求优于难实现的需求

以上四条标准只能定性地从某个角度判断需求的优先级高低。为了对需求优先级实现定量的评估，我们进一步制定了四类积分原则。对于每一个需求，对照这四类积分原则进行积分，计算出该需求的总积分，积分越高的需求优先级越高。

这四类积分原则展示如下：

需求优先级评价角度	需求优先级评价标准	积分原则
功能的重要性	涉及主要功能	加3分
	涉及次要功能	加2分
	不属于必要功能	加1分
涉及到的用户体量	较多用户的需求	加3分
	较少用户的需求	加2分
	极少用户的需求	加1分
功能的使用频率	常用功能的需求	加3分
	较常用功能的需求	加2分
	不常用功能的需求	加1分
需求的可实现性	易实现的需求	加2分
	难实现的需求	加1分

在对每一条需求计算出其积分后，直接按照积分从高到低对需求进行排序。这里实际上是进行了一次**全排序**，但是由于积分相同或相近的需求比较多，我们依据积分划分了四个分数段，标记为四个**等级**（由高到低依次为P1, P2, P3, P4）。

根据上面的积分规则，任意一条需求的积分范围将落在 4 ~ 11 之内，我们将这个范围均匀地划分为四个区间，作为需求优先级的四个等级。需求优先级积分的分数段划分如下：

积分范围	对应的等级
10~11	P1
8~9	P2
6~7	P3
4~5	P4

对于每一条需求，我们考察其对应issue的内容和Label，依据积分规则判定其在功能的重要性、涉及到的用户体量、功能的使用频率和需求的可实现性这四个方面的得分 `rule1_score`, `rule2_score`, `rule3_score`, `rule4_score`，填入Excel表格中。然后使用 `SUM` 函数计算总分：`I3 = SUM(E3:H3)`；得到每一条需求的总分 `total_score` 之后，再使用 `SWITCH` 函数得到每一条需求对应的等级（P1~P4）：`J3 = SWITCH(I3, 11, "P1", 10, "P1", 9, "P2", 8, "P2", 7, "P3", 6, "P3", 5, "P4", 4, "P4")`。（见 2\_详细积分表.xlsx）

通过上述过程，就可以得到所有需求的优先级等级了。我们一共选取了 54 条需求进行分析，最终得到 P1 等级的需求 9 条、P2 等级的需求 15 条、P3 等级的需求 21 条、P4 等级的需求 9 条。（见 3\_P1~P4 分类汇总表（共四张工作表）.xlsx）

## 2. 需求排序性能评价指标

### 1) Kappa系数

Kappa系数是基于混淆矩阵的计算得到的模型评价参数。计算公式如下：

$$k = \frac{p_0 - p_e}{1 - p_e}$$

其中， $p_0$ 是每一类正确分类的样本数量之和除以总样本

假设每一类的真实样本个数分别为 $a_1, a_2, \dots, a_c$ ，而预测出来的每一类的样本个数分别为 $b_1, b_2, \dots, b_c$ ，总样本个数为 $n$ ，则有：

$$p_e = \frac{a_1 \cdot b_1 + a_2 \cdot b_2 + \dots + a_c \cdot b_c}{n \cdot n}$$

评价标准如下：

系数	0~0.2	0.2~0.4	0.4~0.6	0.6~0.8	0.8~1.0
一致等级	极低	一般	中等	高度	几乎完全一致

用python语言中 `sklearn.metrics` 模块的已有接口进行计算，提取需求排序结果中，每一条需求的 `predict_level`, `level` 两列，计算出目标系数的大小。

```
from sklearn.metrics import cohen_kappa_score
kappa = cohen_kappa_score(y_true, y_pred)
```

### 2) 海明距离

在信息编码中，两个合法代码对应位上编码不同的位数称为码距，又称海明距离。海明距离也适用于多分类的问题，简单来说就是衡量预测标签与真实标签之间的距离，取值在0~1之间。**距离为0说明预测结果与真实结果完全相同，距离为1就说明模型与目标结果完全相反。**使用1) 中相同方法进行计算。

```
from sklearn.metrics import hamming_loss
ham_distance = hamming_loss(y_true, y_pred)
```

### 3) 准确率 (accuracy\_score)

`accuracy_score`函数计算了准确率，正确预测频率的fraction (default)，正确预测的数量 `count(normalize=False)`。在multilabel分类中，该函数会返回子集的准确率。**准确率越接近1说明分类越准确。**

```
from sklearn.metrics import accuracy_score
accuracy_score(y_true, y_pred)
accuracy_score(y_true, y_pred, normalize=False)
```

### 4) 微平均 (Micro-averaging)

微平均 (Micro-averaging) 是对数据集中的每一个示例不分类别进行统计建立全局混淆矩阵，然后计算相应的指标。赋予每个样本决策相同的权重，忽略被分类器正确判定为负类的那些样本，大小主要由被分类器正确判定为正类的样本决定，在微平均评估指标中，样本数多的类别主导着样本数少的类。Micro-averaging方法在多标签 (multilabel) 问题中设置，包含多分类，此时，大类将被忽略。**数值越接近1说明分类越准确。**

```
from sklearn.metrics import f1_score, fbeta_score
f1_score(y_true_p, y_pred_p, average='micro')
fbeta_score(y_true, y_pred, average='micro', beta=0.5)
```

## 五、实验结果

### 1. 需求排序结果

#### ① 属于P1等级的需求：

1. 在用户没有打开任何文件夹作为工作区时，VSCode的欢迎页面应当提供“打开文件夹”的按钮。
2. 应当支持对于打开的编辑器列表进行排序。对于编辑器按照文件名进行字典序排序，可以提高用户在打开大量编辑器时的工作效率。
3. 修复撤销重命名Java文件无效的问题。
4. 应当改进Notebook的界面布局，以免用户错过运行按钮或找不到键盘焦点在Notebook中的位置。
5. 应当添加Python插件的功能，使得用户能够在Notebook不受信任时隐藏输出内容；信任Notebook不应将Notebook标记为脏的；当Notebook不受信任时，应当禁用保存/编辑Notebook的功能。
6. 应当增加在Notebook中隐藏输出内容的功能。
7. 在控制台中，输出内容相同的行应当被折叠在一起，以免过多地冗余信息占用控制台。
8. 在调试模式下，界面底部悬停栏应当显示帮助文本/提示，以方便用户切换到普通悬停栏。
9. 应当支持用户选择其最喜欢的变量并收藏，方便调试器将这些用户关注的变量显示在最显眼的位置。

#### ② 属于P2等级的需求：

1. 当用户在打开的编辑器中进行预定义文件过滤搜索时，应当提供“Universe”下拉列表，并且能够自定义包含/排除globs。
2. 当用户复制/粘贴代码时，语言插件应当自动为源代码文件添加相应的using/import语句。
3. 应当改进VSCode的欢迎页面，包括对于新用户和现有用户的欢迎页面，同时提供关于插件和产品迭代的有效更新信息。
4. 应当提高大量文本输出到Jupyter附带的Notebook时的稳定性。修复大量文本输出导致Jupyter附带的Notebook崩溃的问题。
5. 在Notebook带有VSCode风格的按键绑定的同时，用户可以使用他们已经在文本编辑器中使用的快捷键来操纵Notebook的内容。
6. 应当在差异编辑器中提供自动换行的功能。
7. 应当对于断点过滤器添加条件编辑UI。
8. 应当支持条件性捕获异常断点。
9. 应当支持创建与变量关联的ID，使得用户可以从表达式计算创建的变量和调试器中的现有变量。
10. 应当添加用于控制“runInTerminal”请求的转义参数的属性，作为一种控制转义的机制。
11. 应当向用户提供关于终端的快捷键的提示。
12. VSCode应当引导用户设置git config中缺少的user.name和/或user.email。
13. 如果远程更改被pull到已经重新建立基础的分支中，VSCode应该检测到这种情况并警告用户。
14. 对于Git，在存储库选择器中应当支持可选的分支。
15. 应当将GitHub Sponsors集成到扩展视图中，以便用户指定他们希望将哪些用户添加到“赞助者”列表中。

#### ③ 属于P3等级的需求：

1. 应当改进VSCode本地Notebook的可访问性。修复导航导致的混乱；修复Notebook内部的Monaco编辑器无法访问的问题；修复屏幕阅读器访问出错的问题。
2. 应当创建受信任的工作区。用户能够管理工作区shell的权限；诸如eslint之类的扩展应当在工作区中运行代码之前弹出。
3. 用户应当能够在选项卡上高亮显示文件名。
4. 应当支持自定义产品图标主题。
5. 应当只有在特定功能下才需要用户环境，包括打开终端、生成任务、生成扩展主机。
6. 应当支持用户自定义Notebook的markdown渲染器。
7. 应当修复Notebook中HTML元素输出溢出并在编辑器下显示的问题。
8. 应当支持源文件作为多个TS项目的一部分。
9. VSCode应当采用TS 4.1的最终版本。
10. 应当将当前版本的清除器指示器添加到TS版本选择器。
11. 应当更新 JavaScript 调试器用户文档以进行 js-调试。
12. 应当改善动态启动配置用户界面。
13. 用户在扩展主机中进行调试时，应当支持调用本地日志记录API。
14. 调试器的debug.inlineValues应当支持不区分大小写的变量名。
15. Debug模式下不必要提供StackTraceResponse.totalFrames。
16. 应当完善VSCode中的测试功能。
17. 对于终端，应当能够保存重新启动之前的布局。
18. 应当优化资源管理器的用户界面，以引导用户在commit后将其push到远程设备。
19. 应当使得VSCode具有可信类型兼容性。
20. 应当稳定VSCode在apple silicon上的建设。
21. 应当构造一个电子渲染器进程以沙盒模式运行的模型。

#### ④ 属于P4等级的需求：

1. VSCode应当将其Emmet版本更新到最新的2.2.1版本，以免用户遇到由Emmet旧版本导致的问题。
2. 应当扩大Notebook的可用API，以支持更多针对Notebook的功能。
3. 应当适当提高Notebook中差异编辑器的滚动条的滚动速度。
4. 应当在VSCode中提供全屏输出的功能。
5. 应当支持用于诊断的pull模型（当前诊断通过push模型提供）。
6. 应当改进Web UI的停止的指示（“断点命中”）。
7. 应当改进VSCode的PR流程。
8. 应当将WSL作为VSCode的一项内置组成。
9. 应当将Linux ARM deb, rpm和快照发布到存储库。

## 2. 需求排序性能评价结果

首先进行数据提取，将需求分级结果的原有结果和人工分类结果提取，进行简单的处理便于接下来的计算。运行代码（见 `assess.py`），得出各项评价指标的具体值。

评价指标	值	简要评价
Kappa系数	0.3415	一般
海明距离	0.5000	中等
准确率	0.5000	中等
微平均 (f1_score)	0.5000	中等
微平均 (fbeta_score)	0.4444	一般

## 六、结论

### 1. 总体评价

综合代码运行值与评价指标的评价标准进行对比可知，通过本实验所用的分类方法进行处理后的需求排序结果，与原有的分级结果相比，总体可以达到0.5左右的准确率。其中：Kappa系数可达到一般的准确标准，海明距离也居于中等水平。总体来说本实验的分类方法是较有效且有一定依据的方法。

### 2. 误差分析

可以看到，我们的实验中计算出来的结果与真实值之间仍然存在一些差距，分析可能的误差原因如下：

1. 所选需求为VSCode维护团队在2020年11月的工作中涉及到的需求，而非在所有需求中随机选取。所选需求具有一定的主观性，代表性不够强。
2. 由于具有精确等级标签的VSCode需求样本难以找到，本实验用近似的标签代替等级标签进行实验，但这些标签并不够精确。
3. 需求评价中制定的评判标准可能不够全面，未综合考虑到所有相关因素。
4. 需求评价中人工打分的分值具有一定的主观性。

### 3. 实验反思

在软件开发过程中，对需求进行优先级排序有助于软件开发维护团队合理分配有限资源，在短时间内实现尽可能多的重要需求，提高软件开发维护的效率。

在需求优先级排序过程中，我们应当综合考虑多种影响因素，尽量减少人的主观性，提高排序结果的准确性。

## 七、参考文献

- [1] sklearn中的模型评估 [https://blog.csdn.net/weixin\\_42094614/article/details/80117965](https://blog.csdn.net/weixin_42094614/article/details/80117965)
- [2] 【机器学习理论】分类问题中常用的性能评估指标 <https://zhuanlan.zhihu.com/p/30953081>
- [3] 机器学习：多分类模型评价准则 <https://www.cnblogs.com/SakuraYuki/p/13341468.html>