软件需求工程-实验报告

Lab1：软件需求的抽取与分类

2019.10

目录

一、 小组成员

学号	姓名	成绩分配比例
161220056	敬舒舒	0.25
161271030	张梦窈	0.25
161290019	吴雨昕	0.25
171860519	李培剴	0.25

二、 数据获取

本次实验选用 IDE 项目为 Jetbrains 的 Intellij，将该项目的 pull-request 信息作为需求数据，从这些数据中挖掘分析需求。

项目地址为：https://github.com/JetBrains/intellij-community

这里选用的是 pull-request 的标题信息，因为开发者在提交分支时往往会在标题中对这次提交的内容进行概括。而且 pull-request 中提交者与管理员的对话内容大多为问题的具体描述和解决办法，容易混入无关信息且与标题内容重复。

通过爬虫获取到 1130 条 pull-request 信息。

2.1 获取网页数据

利用 pyth on 进行爬虫，我们需要遍历所有 pull-request 页面来获取标题信息。github 的 url 格式较为简单，每个 pull-request 网页的格式为“项目地址+pull/序号”，比如第 100 个 pull 请求的页面为“https://github.com/JetBrains/intellij-community/pull/100”。

需要注意的是，虽然 github 的 pr 序号基本都是从小到大逐次加一，但偶尔会出现某个序号的 pull 请求不存在的情况，因此在爬虫过程中需要判断该次抓取是否成功。

2.2 获取标题信息

通过爬虫我们能得到 pull-request 的网页内容，接下来我们需要从该文本中获取标题内容，这里使用了 bs4 中的 BeautifulSoup 库，利用该库的 find_all 方法找到所有<'span', class_="js-issue-title">的标签，其中的内容便是该 pull 请求的标题。最后将该 pull 请求的序号和标题写入文件

需要注意的是，这里的 pull-request 标题可能是非英文的，对于非英文的内容在写入文件时可能会出现编码错误，这里将出现编码错误的标题写入为 'UnicodeEncodeError'。

三、 需求分析

3.1 关键字分析法

3.1.1 算法思想

关键词分析思路比较简单清晰，首先将爬取下来的 pr_titles 进行分词，每个单词按出现词频进行排序，以此来筛选可能的热门需求。最后将包含关键词的条目依次划分到每个关键词下方，作为该方向的需求内容。

3.1.2 注意事项

直接根据关键词进行词频排序时，常常会看到某些无意义词汇出现频率极高（如 is，in，has，bug 等），为了使获取信息更加精炼更有价值。我们采用了白名单形式，首先在出现频率较高的单词列表中，提取某些有具体含义和实际意义的词汇，列入白名单，之后根据白名单进行过滤，得到有用信息。除此之外，还使用了自然语言处理中的词干提取方法，使用 Python nltk 库中的 wordnet，确保同一单词以不同形式出现时能被归到同一类中。

3.2 LPA 标签分类法

3.2.1 算法思想

标签传播算法（Label Propagation Algorithm）是一种基于标签传播的局部社区划分，半监督学习算法。对于网络中的每一个节点，在初始阶段，LPA 对于每一个节点都会初始化一个唯一的一个标签。每一次迭代都会根据与自己相连的节点所属的标签改变自己的标签，更改的原则是选择与其相连的节点中所属标签最多的社区标签为自己的社区标签，这就是标签传播的含义了。随着社区标签不断传播。最终，连接紧密的节点将有共同的标签。从基本思想来看，它与 Kmeans 算法比较类似，都是在迭代过程中更新当前所属划分，且更新思

想都是取邻居节点中最多的部分。但它无需像 Kmeans 算法一样在初始时即指定聚类数量，而是在传播过程中自动划分与更新，相对更为灵活。

3.2.2 伪代码

```
input：不同需求的节点及它们之间的关系
output：关系划分
1：为每个节点初始化一个标签值。如节点 1 对应标签 1，节点 i 对应标签 i
2：iter=0
3：while true do
4：  对每个节点，统计其所有邻居节点的标签，将其中权重最大的标签赋给当前节点;
5：  if iter≥maxIter or 所有节点的标签都不再变化 then
6：   break
7：  end if
8：  iter+=1
9：end while
```

3.2.3 实现思路

将每个需求视作一个节点，节点之间的链接通过共有的关键词数目来确定。共有的关键词越多，链接越强，比重也就越大。

1) 预处理

预处理过程对原文本进行格式化，去掉了无用的标点符号。得到 id+关键词 1|关键词 2|关键词 3……这样的格式。实现方法是对每一行标题使用正则表达式把匹配到的标点符号替换为空格，再进行 split()，再进行"|".join，以|分割单词列表。

eg：25 IDEA|95644|Support|console|input|filtering

2) step1

上一步得到了标题的单词后，step2 进一步对标题包含的 word 进行提取，以关键词为索引，构建出倒排索引表，统计出每一个关键词存在的所有 pr 的 id。

eg：make 出现在了 6 30 58 104 129 486 518 569 652 915 995 1107 1126 的 pr 里，就把 6 30 58 104 129 486 518 569 652 915 995 1107 1126 这个列表存到一行里。

ps：后来在结果中发现了一些没有实际意义的词出现次数过多，于是我手动删去了 IDEA in to for 四个频率过高且没有实际意义的词的结果。

3) step2

从上一步得到的关键词出现结果中，进一步统计出任意两个 pr 之间共有的关键词个数。这一步及后面的 step3 和 step4 都用到了 pySpark 进行并行计算处理。这里统计结果中的 key 是两个 pr 的 id 号，value 是共有关键词个数。

eg：某行数据是('99,102', 5)，说明 99 号 pr 和 102 号 pr 有五个 word 相同

4) step3

从上面的键值数据中进一步统计对于每个 pr，其余与之相连的 pr 的链接强弱。统计方式是以该 pr 作为 key，计算每个与之相连的 pr 的相同 word 数在所有 pr 的所有相同 word 数中所占比例，把这一系列值作为 value。key 和 value 写在一行里。

eg：对于 id 为 79 的 pr，与之相连的 pr 及链接权重

('79','82,0.14285714285714285|78,0.2857142857142857|696,0.07142857142857142|……

5) step4

第四步是迭代的过程。

首先为每个节点赋予 label 值（也就是本身的 id），然后使用 parallelize 函数创建 RDD，调用 map 转换为 key-value。生成 graph 保存链接关系图。在读入上一步结果的同时也保存了 idList。随后不断基于此关系计算对于每个节点，其邻居节点中比重最大的 label，把这个 label 的值赋给该节点。

eg：某行数据是 240 751,240,481，说明以 240 为标签的 pr 有 751，240 和 481。初步判定它们是一类。随后迭代过程中会再不断更新

ps：由于本实验中 LPA 算法的划分不稳定，所以没有将迭代终止条件设置为节点标签达到稳定，而是在十次迭代中每次都汇总该次迭代的结果，最后再比较哪次的结果比较好。

6) 数据总结

最后读取原始 pr_title，将上面得到的 id 号和文字对应起来并进行输出。十次迭代对应十个 result 文件。最开始都是初始标签，所以前几次迭代的划分结果的种类也比较多。随后不断更新标签，到十次以后只有两个大类又分得太过笼统了，所以理想的划分还是在中间几次当中，需要进一步人工分析哪一次比较好。该方法相比于简单的关键词分类法的改进主要在于不仅基于某个关键字划分，在传播过程中可能会将一些其他关键词进行传递达到统一，但是目前没有比较好的评判标准。

四、 需求分类

4.1 关键词分类法角度

部分高频词汇出现次数如下：

keyword	frequency	keyword	frequency	keyword	frequency
fix	222	remove	40	console	22
add	145	update	36	method	21
support	62	type	34	code	19
use	59	test	33	improve	19
file	50	project	30	python	19
make	42	build	30	dialog	18
class	42	change	30	check	18
name	41	import	25	leak	18
allow	41	new	23	option	17

从该表可以看出，PR 内容最多的是 fix 和 add 相关，即以修补和添加功能为主，其余 PR 需求相对较少且较为平均。接下来根据 classified_pull_request.txt（以下简称 txt）中列出的 PR 具体内容选取部分典型的分类进行说明:

- **fix:** 从 txt 中可见单词 fix 后多跟出错的功能或部件名，也会直接点明某 bug, error 和 warning。可以认为要求进行修补工作是我组采集的 PR 的主要部分。
- **add:** 以添加新功能和对已有功能进行扩展为主。在样本 PR 中出现次数仅次于 fix，且远高于其后的关键词。
- **support:** 表示“提供支持”。值得注意的是 support 多与 add 连用, 且不含 add 的条目也多隐含“添加”之义, 这说明该分类的大部分内容是 add 分类的子集。
- **file:** 文件相关内容。包括两方面：1. 对该工程中的文件处理功能进行完善；2. 对该工程包含的文件进行增删改。
- **class:** 主要涉及程序中“类”的概念。完善对“类”进行处理的功能。
- **name:** 改正工程中的命名错误以及修复工程在处理名称时的 bug。
- **allow:** 开放功能或权限。
- **remove:** 删除冗余或过期内容。
- **update:** 应当说几乎所有 PR 都属于 update 的范畴，但之所以该词频较低，是因为 PR 标题大多会概述更新内容，而使用 update 一词的多是仅指出更新的位置或版本号。
- **type:** 主要是对工程中“类型”问题进行处理。
- **console:** 完善控制台功能。
- **test&check:** 完善工程代码中的检测功能。
- **leak:** 多与 memory 一词关联，显然主要是处理内存泄漏的问题。

- **extension:** 多与 add 一词关联，主要进行功能扩展。
- **window:** 完善某些窗口的功能，如出现较多的 tool window。
- **version:** 注明版本的变更。
- **path:** 多与 fix 一词关联，主要处理路径问题。

4.2 LPA 标签分类法角度

选取 result2.txt 的结果进行分析，以下简称 txt。同样选取部分具有典型性的大类。

- **修复：**修复 bug、内存泄漏、路径错误等问题。
- **添加：**添加和扩展功能。
- **移除：**移除工程中的冗余成分和错误内容。
- **支持：**大多是添加工程对各方面功能的支持，增强功能性。
- **工具窗：**优化工程提供的工具窗的功能。
- **更新：**注明更新了某功能或文件。
- **移动：**将工程中具体的代码的位置进行移动，或移动文件。

4.3 两种分类法的汇总和实例

上述两种方法呈现的分类尽管有出入，但还是具有共性。所以这里两种分类的结果进行汇总，并给出 PR 标题实例。实例包括 PR 标题在原始数据文件中的行号和内容。

1) 修复工程代码中的错误、警告等损害工程功能性和可靠性的点。例如：

55 Fixes IDEA-84220 CTRL-F5 shortcut doesn't work in Hierarchy view!

70 fix commit error on non-ansi Windows and OSX.

86 Fix CoreJavaFileManager bug

157 Fixed a warning, run maven build in maven 3.2.1

168 Fix infinite loop in MavenProjectsTree.findRootProject when there is more than one level of aggregate POMs

2) 增加该工程具备的功能，对已有功能进行扩展，以及提供新的支持。

38 IDEA-36076 Ant integration: allow to add and remove multiple files at on...
113 PY-11175 Add speed search support for Python Integrated Tools
138 add menu item "Navigation" -> "Scroll from source"
1048 Add support for Fish completions in terminal
1150 Add explicit types to method parameters intention

3) 移除工程中无用的代码、模块、功能等。

67 removed duplicate class IElementType.Predicate (use com.intellij.util.containers.Predicate)
408 Remove a special Windows-only call to WinPtyProcess.destroy
1036 Remove dead code from DebugProcessImpl
1129 remove duplicated documentationProvider

4) 对工程文件进行调整，或是对工程的文件处理功能进行改动。

302 Fix file length check
333 Scan files on more threads.
403 Preserve original file when performing copy for postfix template
1168 Fix unstable order of dependencies and modules listed in iml files after import from Gradle

5) 解决工程中内存泄漏的问题。

453 Fix memory leak inside the Gradle daemon caused by Idea Gradle plugin
1088 Fix a memory leak in InternalDecorator (IDEA-208847)

6) 提搞代码的检错能力。

28 optimize plugin version checking when update intellij platform version
238 check myAssertThreading before using assertIsDispatchThread

7) 完善工具窗。

137 add ghost mode to tool window properties (IDEABKL-3654)
161 IDEA-120753: Modified tool window weight to be configurable in the Intellij registry
255 IDEA-57472 new tool window mode using frames