研究问题：自动定义题目难度

代码开源地址：<https://github.com/caposerenity/Data_Science>

研究方法：

考虑到每位同学在AC能力、代码质量、用时、作息时间、编程习惯等方面存在巨大差异，我们希望能根据200位同学的提交记录和提交代码，为他们分别画像。

在开始处理学生编程特点和能力前，我们首先需要对大家完成的题目难度、分类进行处理。就自定义题目难度这一问题，我们使用了test\_data这一较全面的数据集，在下载所有测试数据集后，我们首先抽样查看了约一百份不同题型题目、不同同学的提交代码，发现其中存在如下问题：

·使用C++等python外的语言提交

·面向测试用例编程等刷分现象。

·相同代码反复提交

·没有提交记录却最终满分

为此，我们在评估题目难度之前首先对数据进行了预处理，清洗了面向测试用例、非Python语言的提交的分数记录、异常的满分情况。

得益于提交记录中properties文件记录了该次提交使用的语言，我们很容易判断并处理非Python语言的提交记录，对于非Python2/3的提交，我们将其分数置为0，并对应处理该用户该题目的最终得分。在后续的时间空间复杂度、逻辑复杂度（圈复杂度）等计算时我们也不再考虑这类代码。

在预处理筛出面向用例提交记录中，我们从手动筛选出的80次提交记录的样本集中，选取40例作为数据集，剩余40例作为测试集。方法上，我们采取了从数据集中做特征提取再采用正则表达式等手段做特征匹配方法。针对40例抽样中出现的面向用例特征进行提取，主要包含了如下情况：

·连续的if()print（）；else if（）print（）的情况

·直接以if（）print（）开头且代码长度短于阈值（或显著低于该题目普遍水平）的情况

·仅有print（）内容且print中内容未包含运算的情况

通过对这些行为进行一定的赋分，达到阈值即认为存在面向用例的作弊行为，将这次提交标记为【cheat】并将分数改为0，同时在该同学这道题目的case记录下增加cheat标签。在对挑出的另外20次面向用例提交和30次正常提交记录进行的测试中，我们实现了95%的面向用例检测率和0%的误诊率。基于此，我们对test\_data中的所有提交记录进行了面向用例和是否为python代码的预处理过程。

数据预处理结束后，我们考察了每道题提交的平均分数，20分以下占比，满分占比，平均debug时间和提交次数，综合量化得出每道题目的难度系数。

对于每道题的分数，我们发现在经过预处理后的数据集中，出现的分数有大量的0分和大量的满分，因此仅仅以平均分作为判断难度的依据有失偏颇。因此我们另外综合了每道题的AC率（满分率），1A率，与平均分一起作为分析分数的数据。

在分析分数的基础上，我们通过考察了有关时间的一系列因素，包括debug时间以及提交次数，来得出每道题的最终难度系数。对于debug时间，我们提取学生首次提交和最后一次提交的时间跨度作为考量特征，考虑到时间间隔内不全为debug行为，我们将权重设置为非线性分布，并因这些时间特征具有比较大的偶然性，我们将时间的基础分设为由分数分析得到的基础难度系数，得到最终难度系数

final\_score = 基础难度系数+（基础难度系数\*debug时间权重+基础难度系数\*提交次数权重）/有效的特征数量。

在进行时间分析时，我们认为标记为【cheat】的时间方面记录不具有难度参考价值，因此不做分析。下附权重表。

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 提交时间跨度/min | t=0 | 0<t<=10 | 10<t<=30 | 30<t<=90 | 90<t<=300 | 300<t<=600 | 600<t<=1440 | t>1440 |
| 权重系数 | 0.8 | 0.4 | 0.6 | 0.7 | 0.75 | 0.85 | 0.9 | 1 |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 提交次数 | n=0 | 0<n<=5 | 5<n<=10 | 10<n<=15 | 15<n<=20 | 20<n<=40 | n>40 |
| 权重系数 | 0.8 | 0.5 | 0.6 | 0.7 | 0.8 | 0.9 | 1 |

基于此，我们对分数和时间上的数据分别进行偏态量化（具体过程见附件），得出了每道题目的难度系数。

为了在分析学生编程能力和特征时有更多维度的参考，我们以测试用例作为输入，运行了筛除了面向用例和C++、Java等语言后的满分代码，并获取其耗时情况和内存占用情况，同时我们根据提供的各题标准答案多次运行的平均耗时和内存使用（剔除了异常和极端情况）作出该题的标准资源使用情况。得到学生满分提交记录多次运行后的耗时、内存占用情况之后，我们也和处理标准答案一样剔除了其中的异常值，并对同一题目下各人的时间复杂度、空间复杂度进行了排序处理。

除了时间复杂度和空间复杂度外，我们还统计了学生提交代码中代码行数、注释数量、不规范命名（拼音使用、不含下划线的连续全小写单词等）、引用包的情况，并和时间空间消耗的处理一样，对于代码行数、注释数量进行了排序处理。

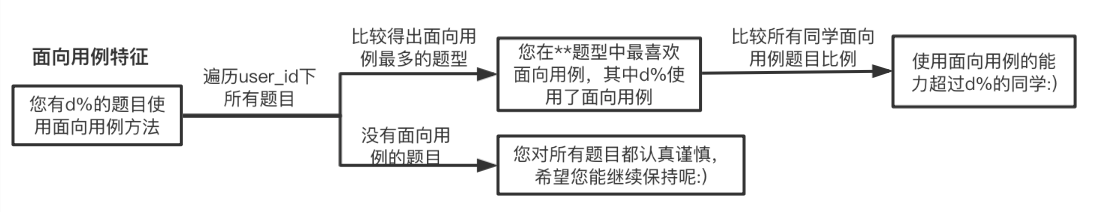
同时，我们考虑到时间复杂度、空间复杂度和长度并不能完全反映一位同学的代码质量。我们这里还引入了圈复杂度，通过计算并统计满分代码的圈复杂度，我们得到了衡量所写代码逻辑复杂度的量化指标，圈复杂度过高的代码也被认为存在暴力解题等问题。

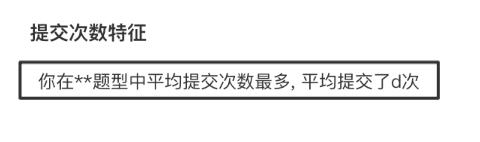
* 学生特征分析

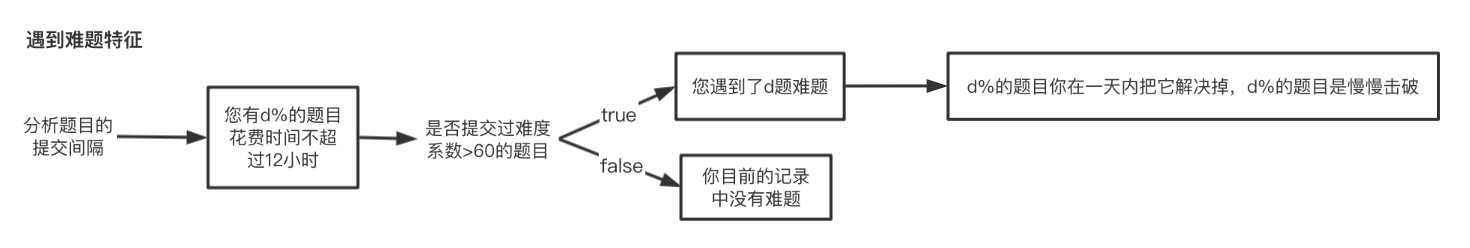
首先，我们考察学生对某类题目的掌握程度。但由于每道题目的难度有差异，因此直接用分数评判掌握程度不太合理。因此我们借助了偏态量化模型[2],根据学生在该题中的位次排名及题目的综合难度量化得出一个偏态量化分数，并综合他是否一次通过（第一次提交即满分），debug用时（第一次提交和最后一次提交的时间差），圈复杂度，时间复杂度再次进行偏态量化，以此结果来判断学生对该题目的掌握程度（具体偏态量化过程见附件）。

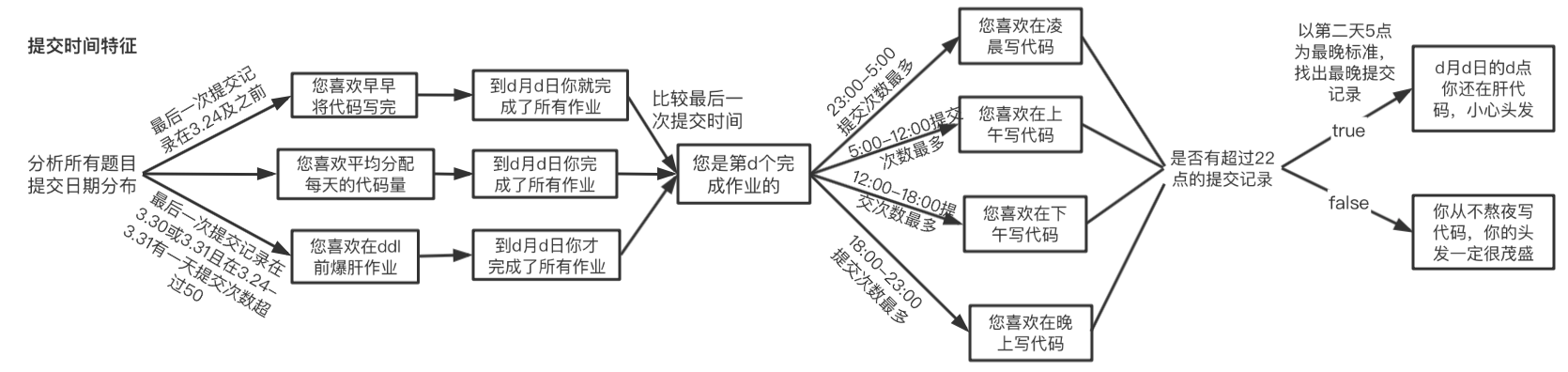
我们希望通过分析题目的提交时间、一道题目的提交间隔、面向用例题目数，得到学生做题的习惯特征，并反馈给学生，以此让同学能清楚地了解自己的做题习惯并希望他们能在此基础上养成良好习惯。

学生特征分为四个方面：面向用例、提交次数、做题时间的特征分析和遇到难题的特征分析，详情如下图。

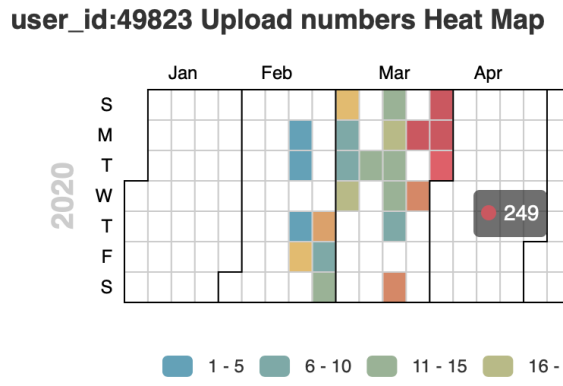
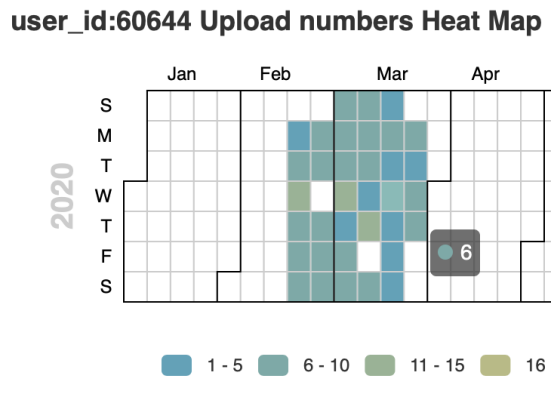
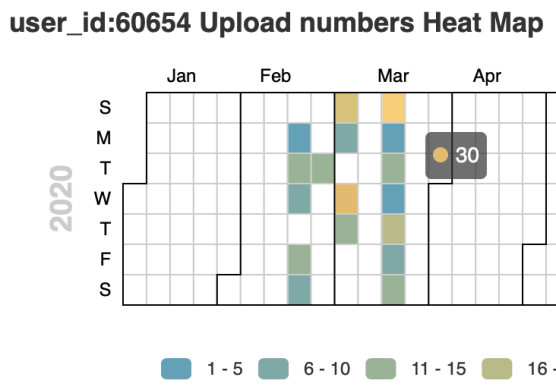




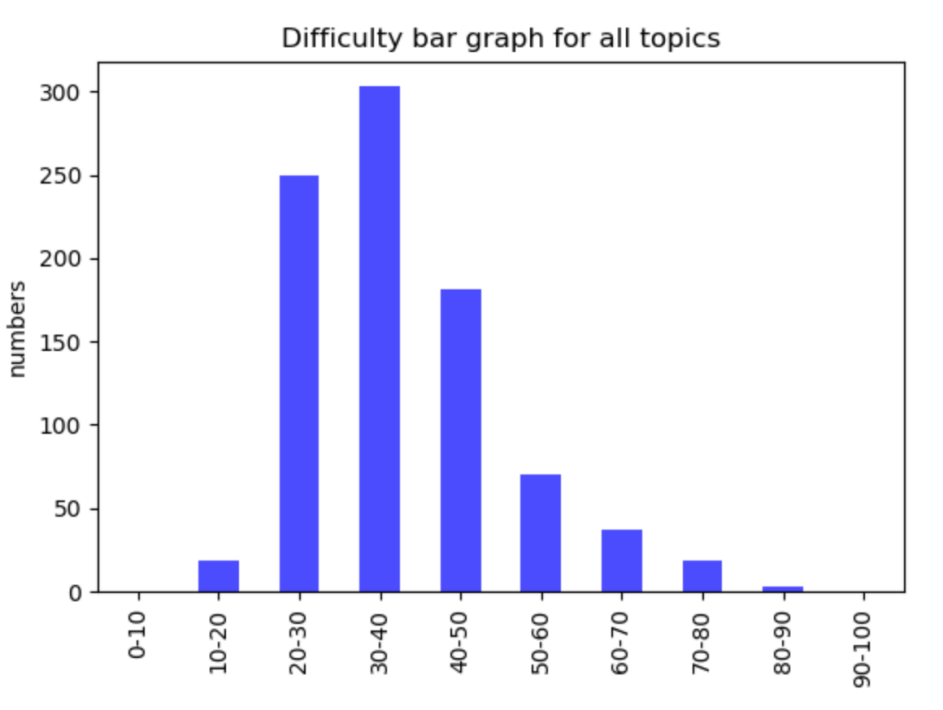




其中，在分类提交时间特征时，我们利用提交时间热力图（src/visual/upload\_num\_heatmap），分类成“早早将代码写完”“平均分配每天代码量”“ddl前爆肝作业”三种不同模式（如下图），如果学生最后一次提交记录在3月24日及之前，判定为“早早将代码写完”；如果最后一次提交记录在3月30日或3月31日且在3.24-3.31中有一天提交总次数超过50判定为“ddl前爆肝作业”；其余的学生类型就判断为“平均分配每天代码量”。



在确定是否为难题时，我们也通过对所有题目的难度掌握（如下图）得出了一个划分难度系数60分，即难度系数超过60就视为难题。



**参考文献**

[1]刘立，王明艳，顾宝程，基于模糊层次分析法的软件编程能力评估模型，吉首大学学报（自然科学版），第３９卷 第２期， ２０１８年３月

[2] 李光兴、余洪琼，考试成绩转换成偏态分布量化成绩的算法，高师理科学刊第40卷第一期，2020年1月

考虑到题目原本的数组、线性表等分类过于粗粒度，我们分析了readme中的题目来源，发现主要源自leetcode、codeforces、geekforgeeks、luogu等OJ网站。而OJ网站上有着对于题目更为详细的分类标签、特点标注（下图是leetcode中部分题目标签展示），严谨性和科学性都超过题目原有的八类分组。



因此我们使用爬虫从上述几个网站中爬取了相关来源题目的标签信息和网站标注难度信息（详细的标签信息可以在tag目录下的tag.csv文件中查看）。



完成题目标签的爬取后，我们分别统计了各个标签的出现次数、出现频率，每位同学对于不同标签的正确率（以上数据可以参考tag目录下personal\_tags.json,tags.json,tag\_type\_count.json三个文件），并且通过可视化手段将每位同学的各个标签正确率加入了该同学的个人情况报告中。此外，为了直观反映约800道题目的知识点覆盖情况，我们根据tag出现频数制作了词云

* 代码风格评估

在考察学生学生python能力时，我们认为学生代码风格也是不可忽视的一部分，上文中提到，我们除了时间复杂度和空间复杂度外，我们还统计了学生提交代码中代码行数、注释数量、不规范命名（拼音使用、不含下划线的连续全小写单词等）、引用包的情况，并和时间空间消耗的处理一样，对于代码行数、注释数量进行了排序处理。

我们利用这些数据来分析学生代码风格评估，得到学生在代码风格上的优点与不足，以供学生改进提高，详情如下图。

