研究问题：学生编程特征画像

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

研究方法：

目录

**数据预处理（面向用例和非python提交）1**

**题目难度3**

评价指标获取3

建立评价指标体系4

**学生编程能力及编程特征7**

学生编程能力7

编程特征12

代码风格16

题目标签16

**学生编程特征画像展示18**

* 数据预处理（面向用例和非python提交）

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

·使用重定向等手段获取测试用例

·完整代码后输出时有部分面向用例手段解决特殊/较难通过的用例

·mooctest测试用例内容出现在代码中

·代码量显著低于该道题目提交记录和本题用python所写标准答案行数

通过对这些行为进行一定的赋分（如下表所示）

|  |  |  |
| --- | --- | --- |
| 情况 | 示例 | 赋分 |
| Print内容为常量/if else后直接接print | Print(“123”) | 每次+1 |
| Mooctest测试用例文件内容完整出现在代码中 | If(in==’1’):  Print(‘123’)  … | 直接判定为面向用例 |
| 大量逻辑后出现print常量内容 |  | 每次+1并用print内容检索mooctest测试用例 |
| 代码量显著低于该题平均提交记录 | 代码量低于平均行数和标准答案的30% | 赋分增加50% |

，达到阈值即认为存在面向用例的作弊行为，我们通过在数据集上进行实验调整，最终将阈值设定为5。对于最终被判定为面向用例的代码，我们将这次提交标记为【cheat】并将分数改为其实际成绩（原有分数-面向用例数/测试用例数\*100），同时在该同学这道题目的case记录下增加cheat标签。并且我们对于该同学本题的实际最终得分也进行了相应的更新。

在对挑出的用作测试数据集的另外40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98%的面向用例检测成功率和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。

* 题目难度
* 评价指标获取

数据预处理结束后，我们考察了每道题提交的平均分数，AC率（即满分率），1A率（第一次提交即满分），平均debug时间和提交次数，综合考量并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，且这样的分数分布不便于使用正态/偏态分布进行拟合。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难度中分数元素的数据。

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。

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。

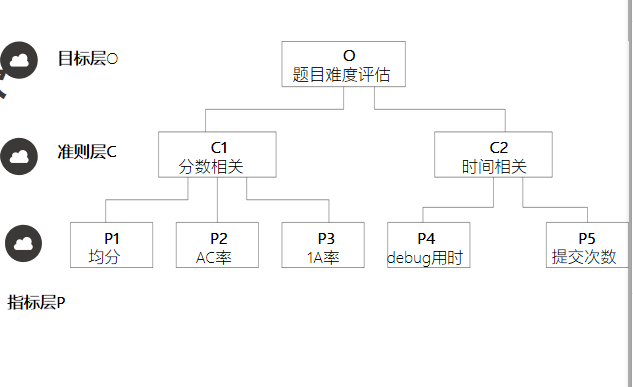
|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 提交时间跨度/min | t=0 | 0<t<=10 | 10<t<=30 | 30<t<=90 | 90<t<=300 | 300<t<=600 | 600<t<=1440 | t>1440 |
| 权重系数 | 0.8 | 0.4 | 0.6 | 0.7 | 0.75 | 0.85 | 0.9 | 1 |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 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 |

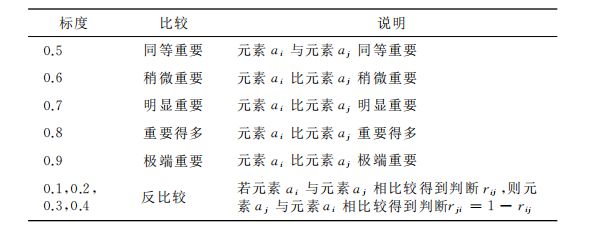
基于此，我们完成了对分数相关数据和时间相关数据的处理。下面我们用模糊层次模型[1]计算每个评价元素在最终难度量化中的占比，具体过程如下：

* 建立评价指标体系

统计分析了各道题目的平均分，AC率，1A（一次通过）率、学生debug平均用时和平均提交次数，对学生的编程能力进行评价，编程能力评估指标体系如下所示．



首先根据标度，确定了三个优先关系矩阵（表F1,F2,F3），具体标度和确定过程如下



其中：C1：与分数相关数据（P1:AC率，P2:1A率，P3：均分）

C2：与提交时间相关数据（P4:平均debug时间，P5：平均提交次数）

|  |  |  |  |
| --- | --- | --- | --- |
| O | C1 | C2 | Ri |
| C1 | 0.5 | 0.7 | 1.2 |
| C2 | 0.3 | 0.5 | 0.8 |

表F1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| C1 | P1 | P2 | P3 | Ri |
| P1 | 0.5 | 0.7 | 0.6 | 1.8 |
| P2 | 0.3 | 0.5 | 0.6 | 1.4 |
| P3 | 0.4(0.5) | 0.4 | 0.5 | 1.3 |

表F2

|  |  |  |  |
| --- | --- | --- | --- |
| C2 | P4 | P5 | Ri |
| P4 | 0.5 | 0.6 | 1.1 |
| P5 | 0.4 | 0.5 | 0.9 |

表F3

通过数学变换［１１］（ｒｉｊ ＝（ｒｉ －ｒｊ）／２ｓ＋０．５，其中ｓ为模糊一致矩阵的阶层），将优先关系矩阵Ｆ１—Ｆ3 ，转换成模糊一致矩阵 Ｈ１—Ｈ3．

|  |  |  |  |
| --- | --- | --- | --- |
| O | C1 | C2 | ∑ｒｉｊ |
| C1 | 0.5 | 0.6 | 1.1 |
| C2 | 0.4 | 0.5 | 0.9 |

表H1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| C1 | P1 | P2 | P3 | ∑ｒｉｊ |
| P1 | 0.5 | 0.567 | 0.583 | 1.650 |
| P2 | 0.433 | 0.5 | 0.517 | 1.450 |
| P3 | 0.417 | 0.483 | 0.5 | 1.400 |

表H2

|  |  |  |  |
| --- | --- | --- | --- |
| C2 | P4 | P5 | ∑ｒｉｊ |
| P4 | 0.5 | 0.55 | 1.05 |
| P5 | 0.45 | 0.5 | 0.95 |

表H3

采用排序法［１２］计算指标权重，计算结果分辨率很高，从而使得决策更加科学．各指标权重ωｉ＝1/s-1/2α＋∑rij/sα，其中ｉ＝１，２，…，ｎ，参数α ≥ｎ－１/２ 。α取（ｎ－１）／２可提高排序结果的分辨率．利用 Matlab求得 各因素权值，Ｃ层相对于Ｏ层各因素权重排序为

ωＣＯ ＝（ωＣ１Ｏ, ωＣ２Ｏ ）＝（0.6,0.4），

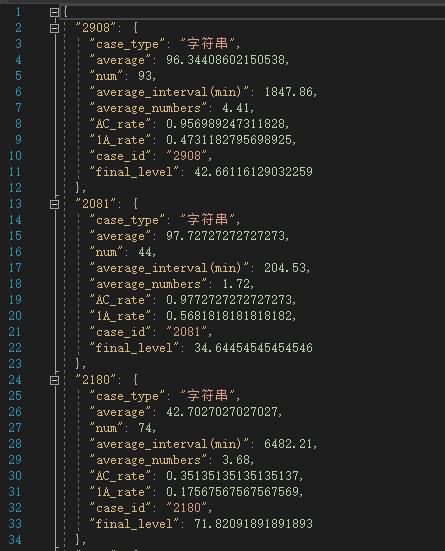
P层相对于C层的权重排序为

ωＰＣ１ ＝（ωＰ１Ｃ１ ,ωＰ２Ｃ１ ,ωＰ３Ｃ１ ）=(0.383,0.317,0.3)

ωＰＣ２ ＝（ωＰ4Ｃ２ ,ωＰ5Ｃ２）=(0.55,0.45)

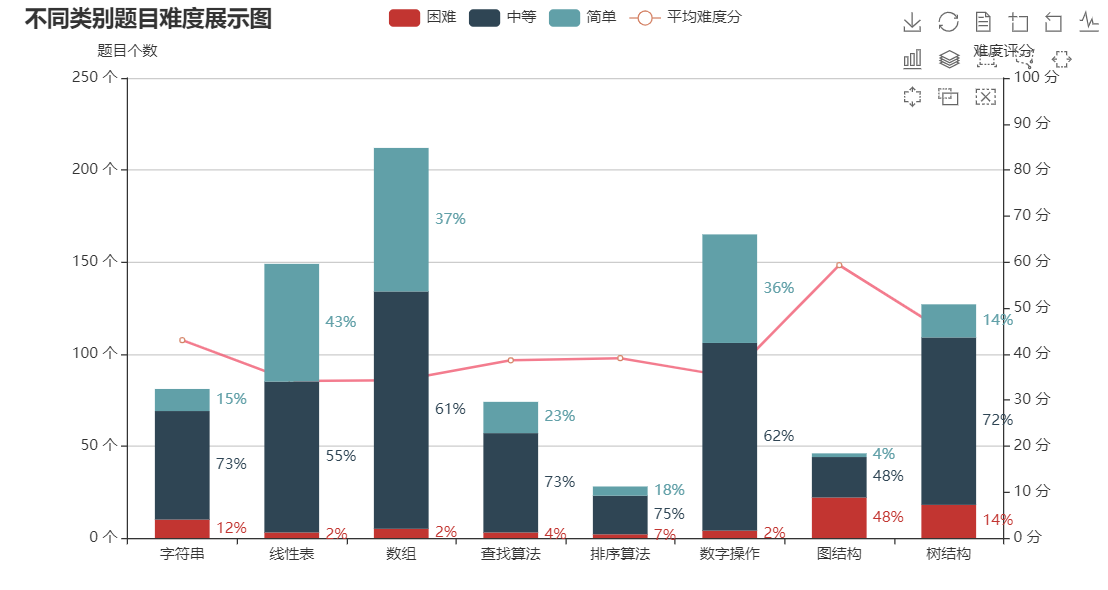
由此得到各因素的难度权重指标。将这些指标代入上文中处理过的评价元素中，即可得到每个题目的难度系数。为了便于理解，我们的难度系数使用的是百分制（即100为最高难度）

最后生成的每道题目难度json文件的部分内容如下，其中“final\_level”即为题目的难度系数



由于确定优先关系矩阵的步骤是我们手工评价和完成，为得到准确的难度数据，我们还按照上述步骤按照三组不同的优先关系矩阵数据进行测试并比较，发现上述优先关系矩阵得到的结果最符合实际情况，由此确定了题目难度。

下图为统计后各个类别题目的难度情况

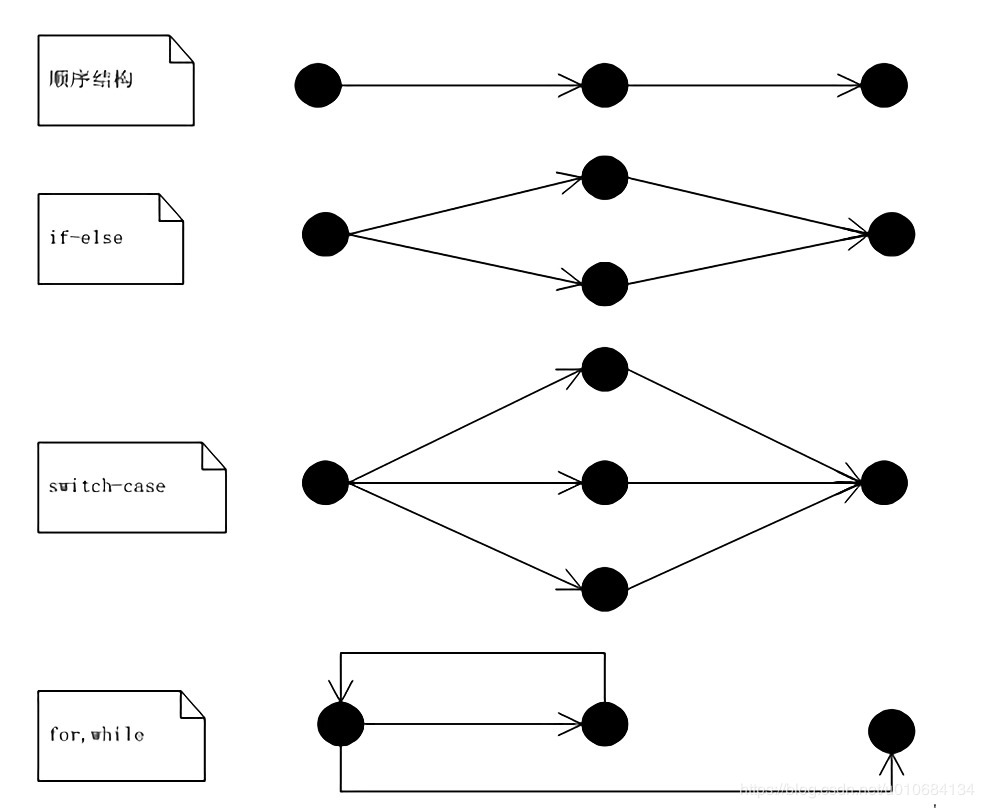


* 学生编程能力及编程特征

由于本次OJ的题目仅有较为宽松的时间限制，很多题目可以通过暴力解法通过，但这一做法显然并不利于学生编程水平提高，我们也不能认为这种情况下学生已经掌握了该道题目的核心内容。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（使用拼音、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，圈复杂度(Cyclomatic Complexity)是一种代码复杂度的衡量标准，由 Thomas McCabe 于 1976年定义。它可以用来衡量一个模块判定结构的复杂程度，数量上表现为独立现行路径条数，也可理解为覆盖所有的可能情况最少使用的测试用例数。圈复杂度大说明程序代码的判断逻辑复杂，可能质量低且难于测试和维护。程序的可能错误和高的圈复杂度有着很大关系。



通过计算并统计提交代码的圈复杂度，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被认为存在暴力解题等问题。同时，对于圈复杂度超过10或者显著（300%）超过该题目平均水平的代码，我们认为其需要进行代码重构。

接下来，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并不合理。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。具体过程如下：

用tk/n表示总体成绩中不超过编号为 k 所对应成绩的数量与总体数量的位次比率。然后，我们将具体步骤简化为

***Step1*** 给出数学期望EX 按照u=100，r=EX-0.5（u\*pi）得到u，r两个关键参数（原文中u为根据方差所得的值，但由于所给数据中方差值实在过大，数据很难处理，所以我们将其直接简化为100）；

***Step2*** 对n 个考生的原始成绩，依次从高到低排列顺序，并对这个排序按 1 到n 进行编号

***Step3*** 对编号为k的成绩，在原始成绩的序列中统计出不超过该成绩的数量，并把这个数量赋值给tk号的成绩

***Step4*** 把tk，n和参数u，r代入转换关系式xk=r-(u\*ln(n/tk))1/2，得出服从偏态分布的量化成绩 xk

把原始成绩转化成量化成绩后，考生成绩的排序位次不会发生变化．由于量化成绩是按照原始成绩在总体中的位次比率关系进行转换的，用相同转换公式转化的量化成绩具有可比性，这样不同学科不同阶段的考试成绩通过转换后变成了可以比较的量化成绩，避免了从原始成绩出发导致的主观臆断错误。

由此，我们通过偏态量化拟合，获得了学生的量化后分数。

但仅仅依靠偏态量化分数来作为评判学生能力的标准依然会出现很多问题。比如，有些学生力求题题满分而有大量提交、有些同学花费了过多时间debug、有些同学用暴力解题导致代码的复杂度非常高，分数不是学生编程能力评估的唯一因素。这些问题在偏态量化分数中并不能体现出来。

因此，在得到偏态量化分数后，我们仅将它作为学生能力评价的主要考量因素中。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，再一次利用模糊层次模型来进行量化，得到学生对某种题目的掌握程度。具体过程如下：

确定了三个优先关系矩阵（表F1,F2,F3），具体确定过程如下

其中：C1：与分数相关数据（P1:AC率，P2:1A率，P3：偏态量化均分）

C2：其他相关数据（P4:平均debug时间，P5：逻辑复杂度排名位次比，P6：时间复杂度排名位次比）

|  |  |  |  |
| --- | --- | --- | --- |
| O | C1 | C2 | Ri |
| C1 | 0.5 | 0.7 | 1.2 |
| C2 | 0.3 | 0.5 | 0.8 |

表F1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| C1 | P1 | P2 | P3 | Ri |
| P1 | 0.5 | 0.7 | 0.6 | 1.8 |
| P2 | 0.3 | 0.5 | 0.5 | 1.3 |
| P3 | 0.4 | 0.5 | 0.5 | 1.4 |

表F2

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| C2 | P4 | P5 | P6 | Ri |
| P4 | 0.5 | 0.7 | 0.7 | 1.9 |
| P5 | 0.3 | 0.5 | 0.5 | 1.3 |
| P6 | 0.3 | 0.5 | 0.5 | 1.3 |

表F3

通过数学变换（ｒｉｊ ＝（ｒｉ －ｒｊ）／２ｓ＋０．５，其中ｓ为模糊一致矩阵的阶层），将优先关系矩阵Ｆ１—Ｆ3 ，转换成模糊一致矩阵 Ｈ１—Ｈ3．

|  |  |  |  |
| --- | --- | --- | --- |
| O | C1 | C2 | ∑ｒｉｊ |
| C1 | 0.5 | 0.6 | 1.1 |
| C2 | 0.4 | 0.5 | 0.9 |

表H1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| C1 | P1 | P2 | P3 | ∑ｒｉｊ |
| P1 | 0.5 | 0.583 | 0.567 | 1.650 |
| P2 | 0.417 | 0.5 | 0.483 | 1.400 |
| P3 | 0.433 | 0.517 | 0.5 | 1.450 |

表H2

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| C2 | P4 | P5 | P6 | ∑ｒｉｊ |
| P4 | 0.5 | 0.6 | 0.6 | 1.7 |
| P5 | 0.4 | 0.5 | 0.5 | 1.4 |
| P6 | 0.4 | 0.5 | 0.5 | 1.4 |

表H3

采用排序法计算指标权重，计算结果分辨率很高，从而使得决策更加科学．各指标权重ωｉ＝1/s-1/2α＋∑rij/sα，其中ｉ＝１，２，…，ｎ，参数α ≥ｎ－１/２ 。α取（ｎ－１）／２可提高排序结果的分辨率．利用 Matlab求得 各因素权值，Ｃ层相对于Ｏ层各因素权重排序为

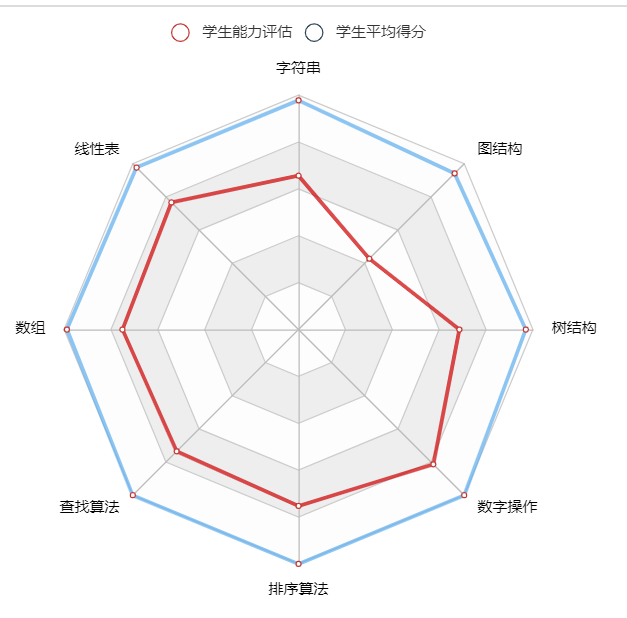
ωＣＯ ＝（ωＣ１Ｏ, ωＣ２Ｏ ）＝（0.6,0.4），

P层相对于C层的权重排序为ωＰＣ１ ＝（ωＰ１Ｃ１ ,ωＰ２Ｃ１ ,ωＰ３Ｃ１ ）=(0.383,0.3,0.317)

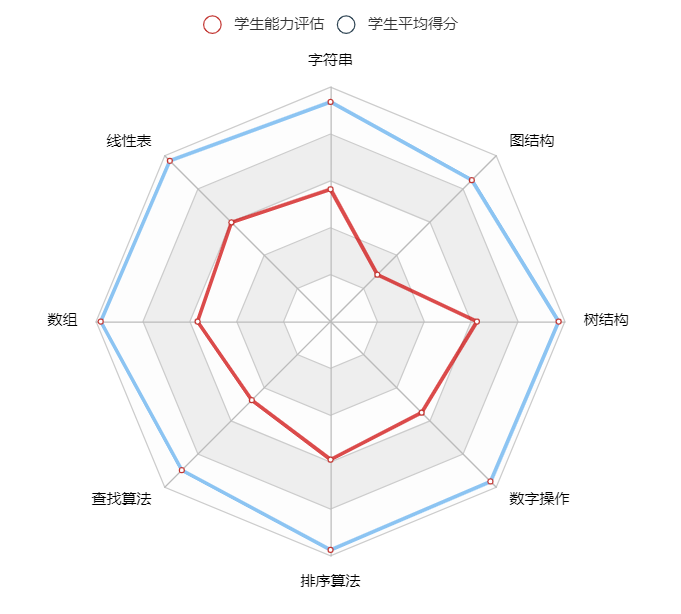
ωＰＣ２ ＝（ωＰ4Ｃ２ ,ωＰ5Ｃ２ ,ωＰ6Ｃ２）=(0.4,0.3,0.3)

由此得到各因素的权重指标。将这些权重代入各个评价元素中，即可得到学生在每道题目的表现。为了便于理解，我们的量化结果同样使用的是百分制（即100为最高掌握程度）

以id为60591的学生为例，生成的他对各类型题目掌握程度如下：



作为对比，我们制作了平均得分与他相近，但综合编程能力上存在明显差异的60618同学的雷达图



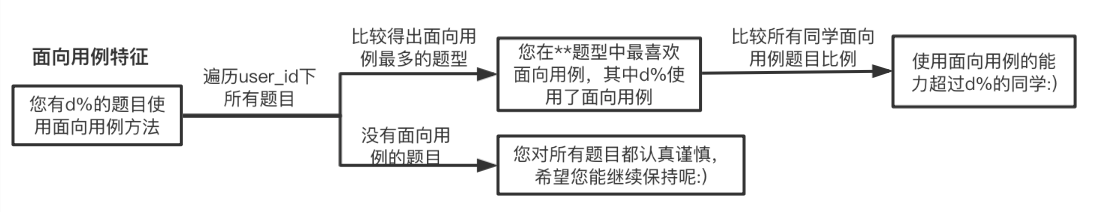
上述对比中我们可以发现，平均分在各类型的题目中几乎均匀分布，且各个同学之间不存在显著差异，并不能帮助学生找出自己的不足。但加入了其他考量因素后，各个类型的掌握程度出现了明显的区别，可以帮助学生更好认清自己的优势和不足。

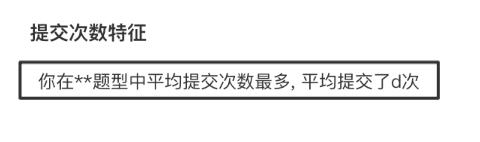
* 编程特征分析

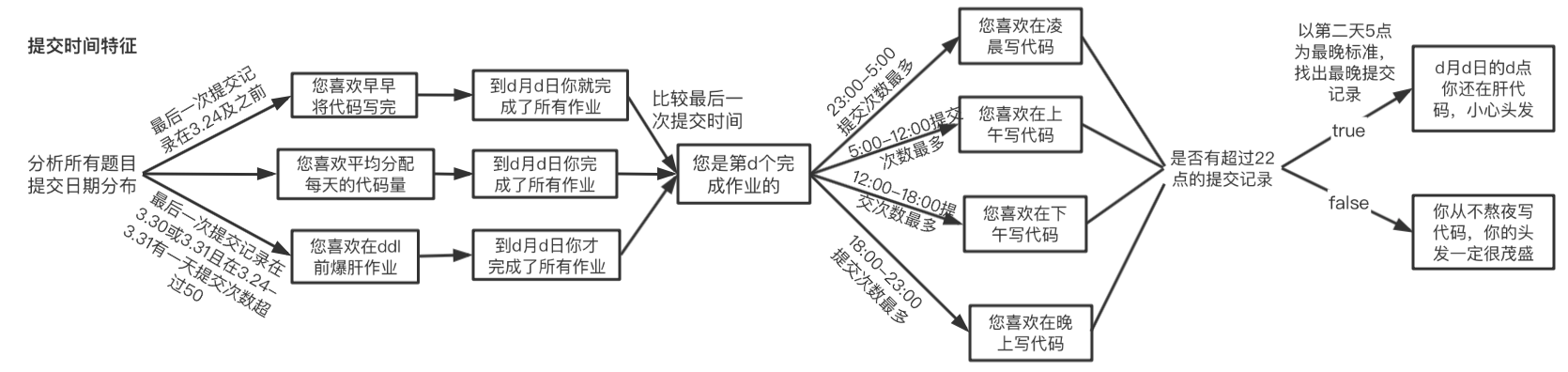
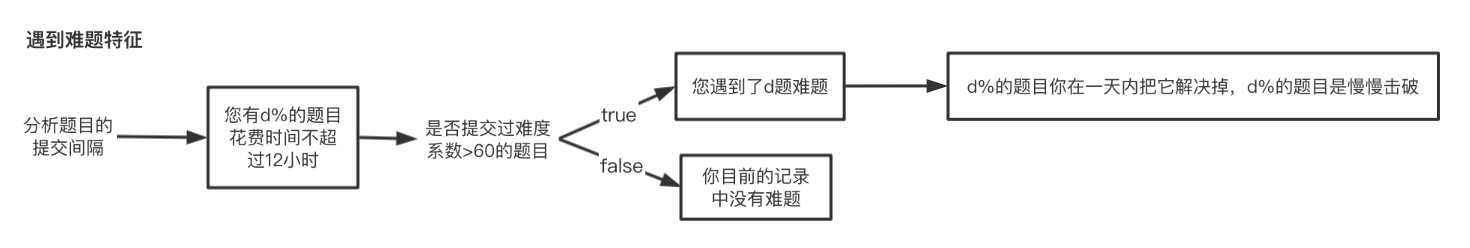
除了编程能力，学生还有其他很多不可忽视的编程习惯于特点，我们希望使用统计学手段分析出这些特点。

我们通过分析题目的提交时间、一道题目的多次提交的时间间隔、面向用例题目数和比例、注释习惯、命名规范性等，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。

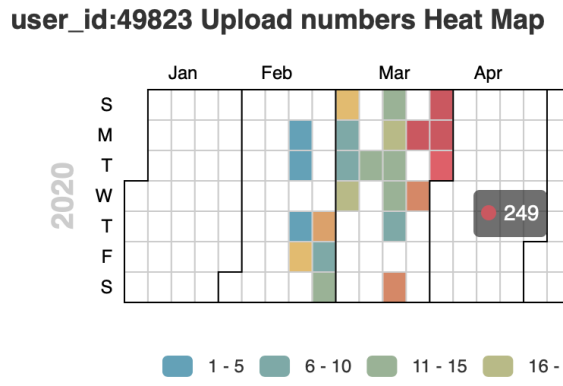
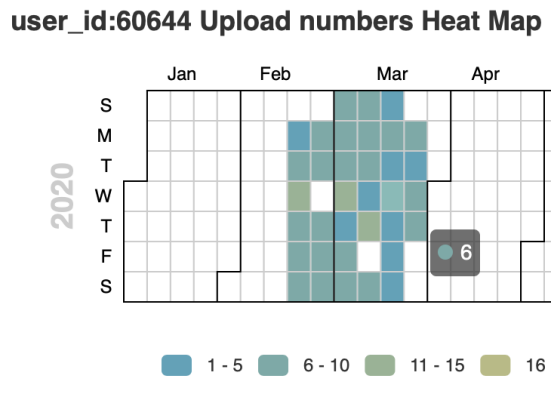
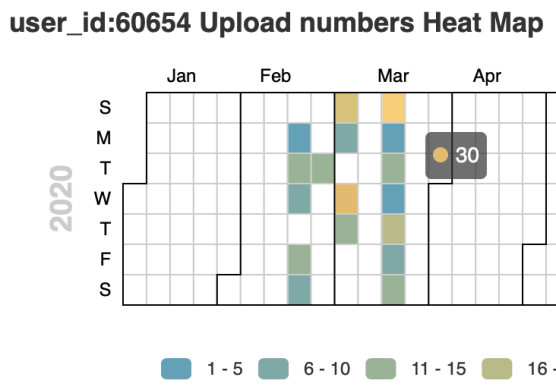
学生特征分为四个方面：面向用例、提交次数、做题时间的特征分析和遇到难题的特征分析，详情如下图。



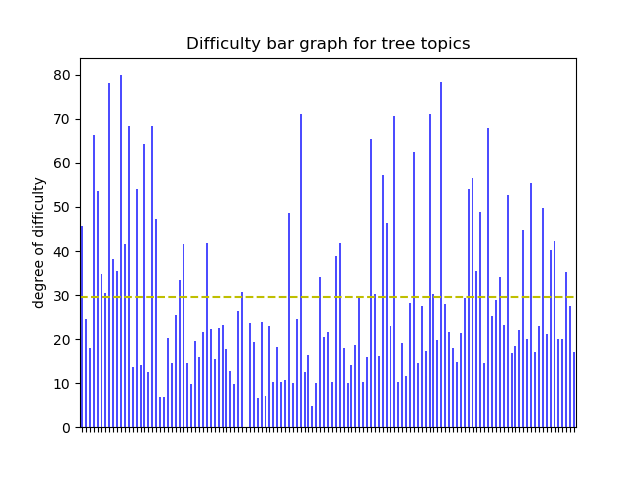


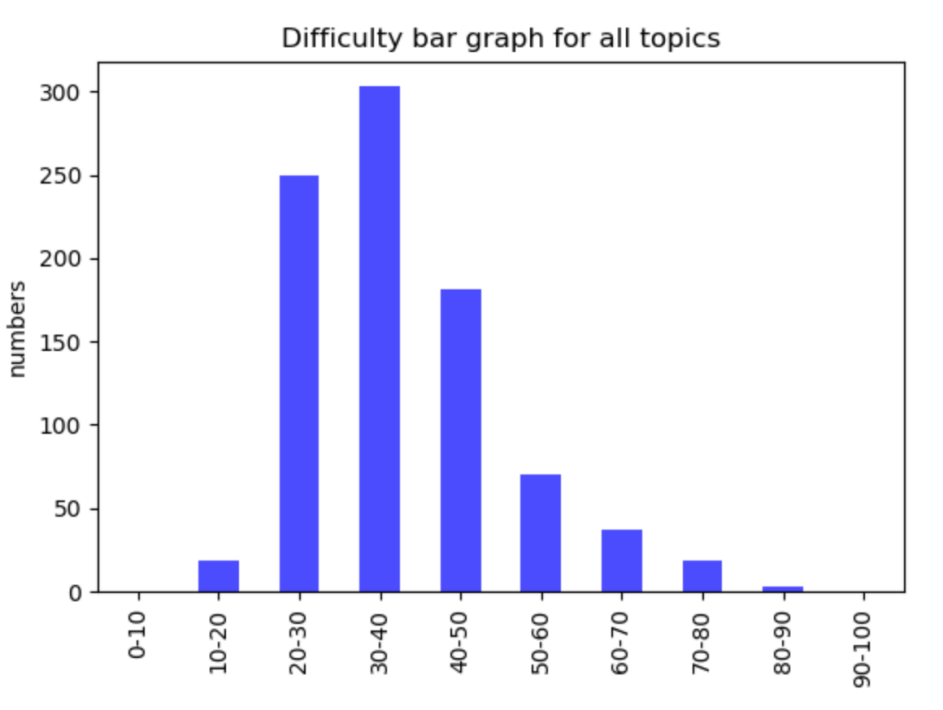


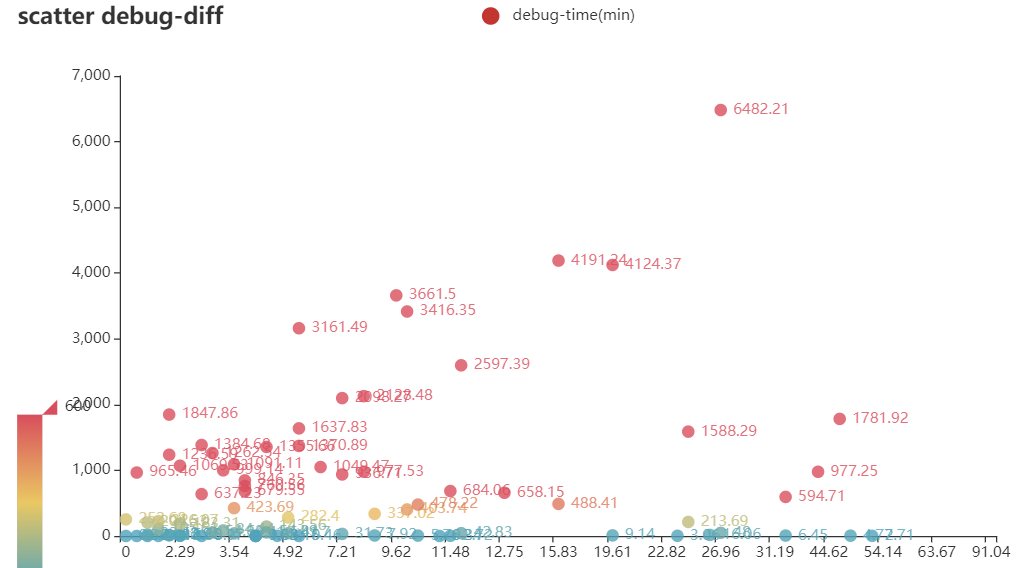
其中，在分类提交时间特征时，我们利用提交时间热力图（src/visual/upload\_num\_heatmap），分类成“早早将代码写完”“平均分配每天代码量”“ddl前爆肝作业”三种不同模式（如下图），通过统计，我们将仅有30%同学全部完成的3.24定位较早完成日期。如果学生最后一次提交记录在3月24日及之前，判定为“早早将代码写完”；如果最后一次提交记录在3月30日或3月31日且在3.24-3.31中有一天提交总次数超过50判定为“ddl前爆肝作业”；其余的学生类型就判断为“平均分配每天代码量”。除此之外，我们还统计了各位同学每天提交的时间分布，并分析其喜欢的写代码时间、每天用时、熬夜情况，并给出类似“3月17日凌晨2:31你还在肝代码，小心头发”等建议。



在确定是否为难题时，我们这里使用了先前模糊层次计算得出的题目难度量化指标，将难度超过60的题目是为难题。下图一二两张分别是树结构的题目难度情况和所有题目总体的难度分布统计。在对于用户面对难题的特征时，我们统计了用户解决该题目花费的时间和最终总提交次数，并分析了该同学面对难题时的偏好（爆肝做完还是先放在一边，最后处理）。而对于难题，我们还提取出了各位同学对不同题目的debug时间，希望使用函数进行拟合，但分布点图并不像预计的那样存在较明显的线性关系（见下图三，x轴为题目难度，y轴为debug用时）







* 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，并对各位同学给出了“时间复杂度普遍过高，尝试优化算法”，“代码长度过长，试试重构”等建议。其他一些不良习惯（诸如中文拼音命名、格式混乱等），我们也都给出了相应的建议,以供学生改进提高。

* 题目标签

考虑到题目原本的数组、线性表等分类过于粗粒度，且很多题目并不能被简单的一个词所概括。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），准确性性和科学性都超过题目原有的八类分组。



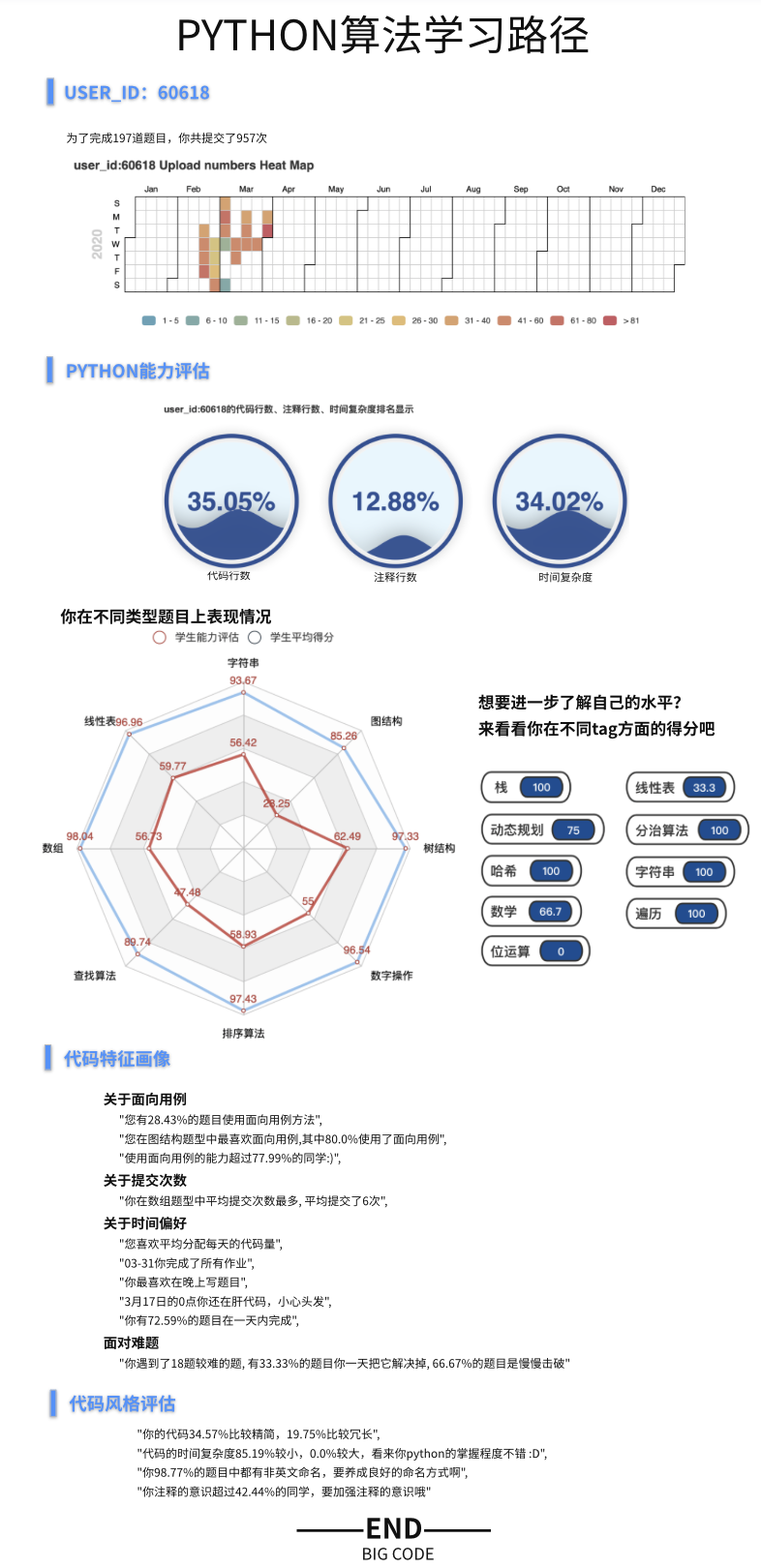
因此我们使用爬虫从上述几个网站中爬取了相关来源题目的标签信息和网站标注难度信息（详细的标签信息可以在tag目录下的tag.csv文件中查看）。



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

在获取了上述tag分类之后，我们将各位同学的各题型得分情况细化为各标签的得分和掌握情况，并且针对该同学掌握情况反映他还需提高的部分。

* 可视化展示



**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

[3] 陈荟慧，熊杨帆，蒋滔滔，李佳，基于在线测评系统的编程题目难度研究，重庆大学计算机学院，重庆大学教学改革研究项目（No.2017Y55）