Python 练习研究报告

小组信息

成员信息

成员人数：3人

学号	姓名	邮箱	Python练习完成情况
181250119	任一丁	181250119@smail.nju.edu.cn	200/200
181250083	林希澄	181250083@smail.nju.edu.cn	199/200
181250168	薛人玮	181250168@smail.nju.edu.cn	106/200

组员分工信息

任一丁：负责对代码的时间/空间复杂度判定的代码编写；负责人工测试和数据统计调查。

林希澄：负责前端页面；后端代码接口架构；代码原创性/相似程度判定的代码编写。

薛人玮：数据处理；代码风格评价的代码编写；前后端连接件的编写。

研究问题

通过对代码原创性、时空复杂度、代码风格水平三个维度的评分评价：

1. 单一学生的某个代码样本，在学生样本集所提供的代码样本集中的优劣程度。
2. 单一学生在学生样本集当中的编程能力高低。

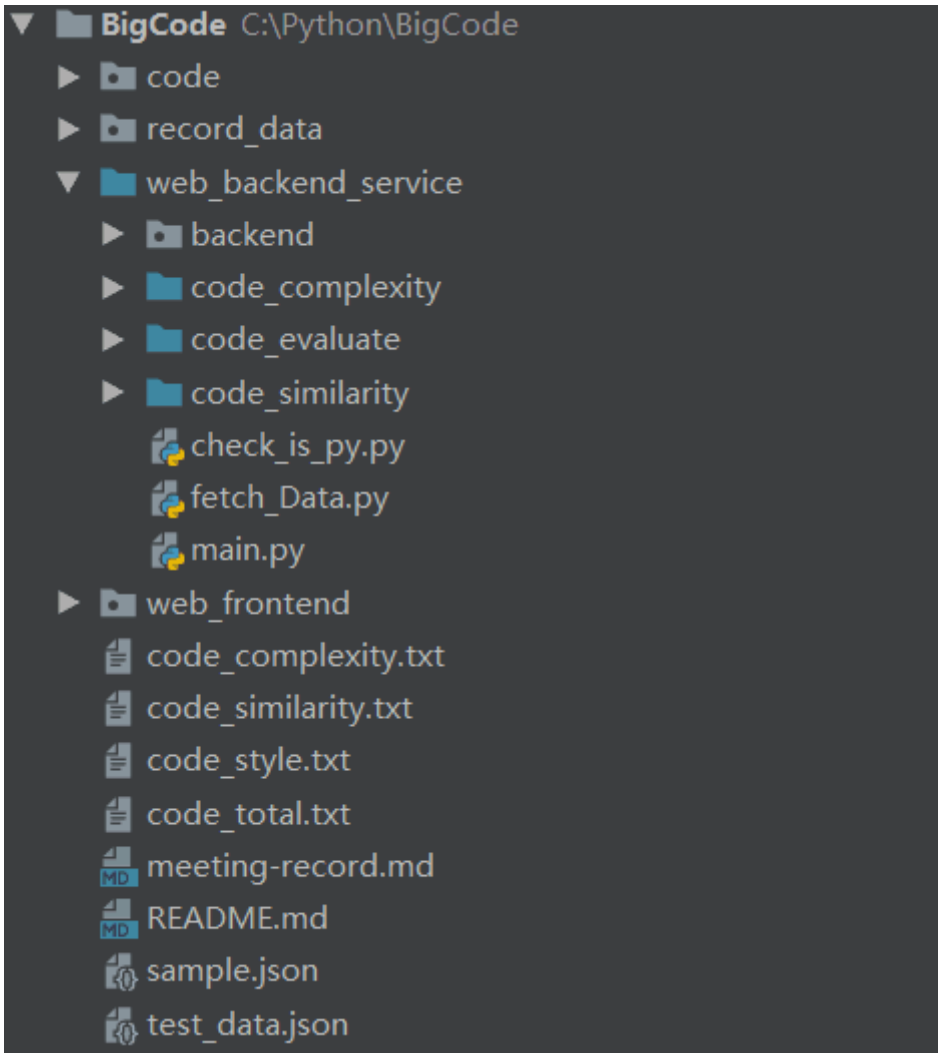
项目应用场景：用于小体量的学生Python算法练习评价。

代码简介

Github开源地址

https://github.com/cclintris/BigCode

代码架构



文件	功能
BigCode/code	存放所有学生所有题目最高得分/最近一次代码相关
BigCode/record-data	存放项目报告相关
BigCode/web-backend-service/backend	前后端连接件/后端接口
BigCode/web-backend-service/code_complexity	代码的时间/空间复杂度判定相关代码
BigCode/web-backend-service/code_evaluate	代码风格评价相关代码
BigCode/web-backend-service/code_similarity	代码原创性/相似程度判定相关代码
BigCode/web-backend-service/check_is_py.py	检查代码是否使用Python3相关代码
BigCode/web-backend-service/fetch_Data.py	处理test_data.json数据相关代码
BigCode/web-backend-service/main.py	分析结果处理相关代码
BigCode/web-frontend	分析结果的前端呈现
BigCode/code_complexity.txt等4个txt文件	存放分析结果相关

研究方法

代码时空复杂度评价

原理：

分析每个代码样本的各项代码评价指标（时间复杂度、空间复杂度），列出这些指标，并与同一题目的样本集作比较，给出评价。通过对单一学生所有代码评价指标的统计，可以得出学生在样本集中的相对水平高低。

限制：

大部分复杂度分析方法的代码根据python语言的特点编写，并且使用纯文本分析方法，不涉及语法树的构建以及语义分析。因此该方法较适用于小型的python算法题（代码行数<50）评价，在逻辑结构复杂、编写方法较多的代码样本中方法判别的错误率会显著升高。

评测：

对提交的42800+份代码进行**等距抽样，**样本距离为100，共抽取428份代码样本进行人工评测。其中符合人工评测期望的机器评测样本数量为402份，机器判别有效率可达94%左右。考虑到设计逻辑上的部分缺陷，实际的有效率会比评测的有效率略低一些。相关统计详情见附录中的EXCEL表格以及对人工评测标准的相关说明。

分析思路：

复杂度分析采用了**文本分析**方法，针对python特殊的缩进式语法结构构建**缩进列表**（缩进树），并利用该列表判断嵌套结构。复杂度分析使用了定位算法题中常见的逻辑结构保留字（for、while），以及常用的算法复杂度语言特征（二分查找、简单尾递归等），使得在小型算法题中可以较为准确地判断循环层数和单一方法的复杂度。

该分析方法的缺陷为：方法会忽略import包产生的复杂度；方法在多个方法嵌套调用的情况下（如间接递归），会忽略来自其他方法的复杂度；分析方法的过程中，循环销毁的变量可能会被误判为递归压栈导致空间复杂度上升。

代码原创性评价

原理：

取得一样本代码，通过分析此样本与属于相同题目的其他所有样本进行**相似度比较**，若是此样本与其中任一样本的相似度评估超过我们对此样本所属题目设定的**阈值**（此阈值通过对**样本答案的难度评估**以及**平均学生得分**得出），则对原创性分数予以相应扣除。通过对单一学生所有代码评价指标的统计，可以得出学生在样本集中的相对水平高低。

缺陷：

在题目集中，有些题目较为简单，解题思路相对固定单一，代码总量相对较少（例如简单的冒泡排序题）。针对这类题目，虽然我们将阈值相应的提高，但由于我们对代码的相似度比较将更改变量名、移动代码块等情况纳入考量，所以可能导致出现阈值提高至100%的情况，从而出现漏判。但鉴于这种题目较为简单，出现这样的漏判显然是可以接受的。

分析思路：

实现代码相似度判别的基本思想要分为两部分进行。第一部分，要对代码进行初步的**文本相似分析**，第二部分，是要对两份欲进行比较的代码(以下简称**引用代码：refcode**以及**候选代码：candcode**)生成各自的**代码语法树**，并比较这二棵代码语法树节点个数以及其相似程度百分比，藉此推算出两分代码的相似度。

第一部分：

我们必须避免 candcode 仅仅搬动 refcode代码片段却也能运行出相同结果的剽窃手法。我们使用了 **K-gram 算法**来实现。K-gram 的主要做法就是将一串文本以一定大小的窗口进行分割，分割的单位称为一个 shingle，且每个 shingle 都是长度为 k 的起始窗口设定大小，最后，对我们的 candcode 和 refcode 都建立好各自的 shingle 集合后，对这二个集合比较重复的 shingle 比例，藉此来判别 refcode 有多少比例疑似是抄袭 candcode 的。显然，这个方法可以很好的避免代码搬动的抄袭手法。最后，k 起始窗口大小取值必须要过滤掉那些在文本中非关键的词汇。基于我们的项目需求，由于要分析的文本对象是 Python 代码，所以像是代码中常见的词汇 *if*，*elif*，*while*，*for*，*def*，*main*等等就属于我们不关心，且不能做为抄袭依据的非关键词汇，应予以过滤。

第二部分：

要生成整份代码的语法树，就要先对代码进行分解。这边，为了统一，我们都在所有代码样本要进行构建语法树之前，我们都先对代码做了预处理，在开头添加一个 *def f():* ，并将原代码增加一级缩进，使该函数成为代码的入口。

接着，我们将代码按行分解，并将所有行存进 *pycode_list* 列表中。接着，构建语法树的过程，首先，遍历 *pycode_list* 并对每个单元进行分析，这边我们引入了 Python3 中的 *ast* 模块，并使用了 *ast.parse()* 来构建第一步的语法分析树。接着，如上面 *K-gram* 算法所述，我们必须过滤掉所谓的非关键词汇，因此我们建立了一个 *FuncNodeCollector* 类。该类的做用在于对未经处理的语法分析树，引入 *K-gram* 算法，清除那些在语法分析树中非关键词汇的节点，并收集剩下对判别相似度有意义的节点，最后返回经过处理的语法分析树的根结点，以利之后遍历该树。

对 *refcode* 和 *candcode* 都构建完经过处理的语法分析树后，接着要进行比较。进行比较这部分，我们引入了 Python 的 *zss* 模块，并使用了 *zss.distance()* 函数。*zss.distance()* 函数可以计算出两棵 *ast* 树的编辑距离。我们也搭配语法分析树的节点总数做为一个判断的依据，最后将所有信息，包括节点数量和两树之间的编辑距离等信息存入一个 *ast_diff_result* 列表。最后再对 *ast_diff_result* 中的结果进行加总求平均，并将该运算结果取小数点二位，最后返回出该值做为 *refcode* 和 *candcode* 之间的相似度。

代码相似度分析的代码部分详见 *code_similarity* 文件夹。主要见 *pycode_similarity.py* 和 *K-gram.py*。*hash.py* 仅为了使代码效率增快，对 *K-gram* 集合建立哈希表，与代码相似度分析并无直接关系。*syntax_tree.py* 参考了 Python3 zss 库原码并应项目需求进行了部分重构。*index.py* 为测试文件。

代码风格水平评价

原理：

分析每个代码样本的各项代码评价指标（如下），列出这些指标，并与同一题目的样本集作比较，给出评价。通过对单一学生所有代码评价指标的统计，可以得出学生在样本集中的相对水平高低。

> 代码是否只是用一种缩进
>
> 缩进时，空格数是否是4的倍数
>
> 单行代码是否没有超过既定长度
>
> 代码是否没有尾随空格
>
> 操作符两边是否有空格
>
> 定义中的操作符两边是否没有空格
>
> 是否只使用了一种引号(特殊情况除外)
>
> 代码开头是否不是空行
>
> 代码是否没有行内注释
>
> 以"#"开头的注释"#"后是否有空格
>
> 在import之后是否有空行
>
> 在class之前是否有空行
>
> 在def之前是否有空行
>
> 是否没有在同一行的import导入不同的包
>
> 用from开头的import是否在其他的import之后
>
> import之间是否没有空行
>
> 变量名命名是否有意义/合理

缺陷：

以上指标是选取**PEP 8**标准中较为核心的17个指标。大部分分析方法的代码根据Python语言的特点编写，并且使用纯文本分析方法，不涉及语法树的构建以及语义分析，故而无法识别出与语法相关的违背**PEP 8**标准的内容。

分析思路：

代码风格分析采用了**文本分析**方法，针对Python特殊的缩进式结构构建**缩进列表**（缩进树），并利用该列表判断嵌套结构。代码分析使用了定位算法题中常见的保留字（*def*、*class*、*import*、*from*、*#*、*'''*、*"""*），同时需要判断此关键字是否处于**有效代码**，对出现关键字的周围内容行数添加进敏感列表。

以下报告顺序与上表一致。

> 当代码中出现了以**\t(tab)**形式的缩进时，就认为此样本代码全篇缩进皆使用**\t(tab)**，当之后检测出任一缩进是使用**space**时，将此指标判为否；反之，亦然。
>
> 当在判断**指标1**时，只有认为样本代码全篇缩进皆使用space时，才会进入此判断。
>
> 在文本列表中，寻找有无长度超过既定长度的。
>
> 在文本列表中，寻找有无代码内容有无意义尾随空格的。
>
> 通过缩进树，判断出现操作符的位子不处于*def*中，进而判断有无环绕空格。
>
> 通过缩进树，判断出现操作符的位子处于*def*中，进而判断有无环绕空格。
>
> 当代码中出现了以**单引号（''）**形式的引用时，就认为此样本代码全篇皆使用**单引号（''）**，当之后检测出任一缩进是使用双引号（""）时，将此指标判为否；反之，亦然。此外考虑有两种特殊情况：1. 双引号单引号混用时，若是避免引号嵌套的歧义，增加代码可读性时，指标判断将不会是否。2. 双引号单引号混用时，若是为了区分字符串与单字符时，指标判断将不会是否。
>
> 在文本列表中，查看文章开头是否有空行。
>
> 在文本列表中，判断单个元素是否是有效代码，在此行是有效代码的基础上，检查有无注释。
>
> 在文本列表中，检查单个元素检查有无以#起头的注释，并检查有无跟随空格。
>
> 定位样本代码中关键字*import*，此关键字是否处于有效代码，将*import*上下行数纳入敏感列表。在此列表中，检查空行情况是否满足PEP8标准。
>
> 定位样本代码中关键字*class*，此关键字是否处于有效代码，将*class*上下行数纳入敏感列表。在此列表中，检查空行情况是否满足PEP8标准。
>
> 定位样本代码中关键字*def*，此关键字是否处于有效代码，将*def*上下行数纳入敏感列表。在此列表中，检查空行情况是否满足PEP8标准。
>
> 定位样本代码中关键字*import*，此关键字是否处于有效代码，将*import*周围元素纳入敏感列表。在此列表中，检查导入的包是否有且仅有一个。
>
> 定位样本代码中关键字*import*，*from*，此关键字是否处于有效代码，将*import*，*from*周围元素纳入敏感列表。在此列表中，检查*from*导入的包是否位于*import*代码块之后。
>
> 定位样本代码中关键字*import*，此关键字是否处于有效代码，将*import*上下行数纳入敏感列表。在此列表中，检查空行情况是否满足PEP8标准。
>
> 通过读取此样本代码的所属题目的*readme.md*文件，将题意综述内容机器翻译成英文，将其中含有题设意义的关键字（排除了is，has等高频词汇）与标准答案之命名进行交叉，建立一个命名库。将样本代码的命名与命名库中进行比对，从而得出对这一指标的匹配值。

在对于检查例如有无导入无用包时，我们将包名定位关键字，判断此关键字是否处于有效代码。在此文本列表中，检查是否有引用关键字（包）并且处于有效代码中。

但是，由于代码风格带有主观性，同时内容繁杂众多，仅仅通过这些维度去描写出一个同学的编程风格水平略显单薄。但是由于上述所选择的指标皆在PEP 8中处于较为核心的位置，所以我们认为，在上述指标皆合格的情况下，样本才有可能是一份风格优秀的代码；反之，则有可提高的空间。

案例分析

研究问题：

通过对代码原创性、时空复杂度、代码风格水平三个维度的评分评价：

1. 单一学生的某个代码样本，在学生样本集所提供的代码样本集中的优劣程度。
2. 单一学生在学生样本集当中的编程能力高低。

研究问题1

此处，使用**user_id：16304**，**case_id：2354**作为案例分析。

case_id："2354"
Snuke 从他的母亲那里得到了生日礼物——一个网格。网格有H行W列。每个单元格都是黑色或白色。所有黑色单元格都是四联通的，也就是说，只做水平或垂直移动且只经过黑色单元格即可从任何黑色单元格移动到任何其他黑色单元格。

第i行第j列（1≤i≤H，1≤j≤W）的单元格的颜色由字符s_{ij}表示。如果s_{ij}是"#"，该单元格为黑色；如果s_{ij}是 "." ，该单元格为白色。至少一个单元格是黑色的。

我们定义「分形」如下：0级分形是一个1 * 1的黑色单元格。k+1级分形由H行W列较小一级的分形按照 Snuke 的网格的样式拼成：与 Snuke 网格中的黑色单元格对应的位置是一个k级分形；与 Snuke 网格中的白色单元格对应的位置是一个单元格全部为白色，尺寸与k级分形相同的网格。

您将得到 Snuke 的网格的描述和整数K。请求出K级分形中黑色单元格组成的连通分量数，模10⁹+7。

原创性评分

将此样本代码与其他所属同一case_id的样本代码进行相似度比较，可得如下数据集。


```
{'16304': {2354': {'40186': Decimal('89.80'), '47920': Decimal('48.98'),
'47937': Decimal('97.96'), '47961': Decimal('95.92'), '48025': Decimal('63.27'),
'48083': Decimal('97.96'), '48721': Decimal('77.55'), '49361': Decimal('89.80'),
'49823': Decimal('36.73'), '52592': Decimal('77.55'), '58547': Decimal('81.63'),
'58575': Decimal('97.96'), '58585': Decimal('97.96'), '58586': Decimal('67.35'),
'58610': Decimal('87.76'), '58616': 'one of them is not python file', '58634':
Decimal('67.35'), '58778': Decimal('100.00'), '58822': Decimal('81.63'),
'59137': Decimal('77.55'), '59140': Decimal('77.55'), '60580': Decimal('85.71'),
'60591': Decimal('85.71'), '60592': Decimal('69.39'), '60593': Decimal('65.31'),
'60594': Decimal('77.55'), '60601': Decimal('87.76'), '60605': Decimal('69.39'),
'60608': Decimal('97.96'), '60610': Decimal('87.76'), '60611': Decimal('65.31'),
'60614': Decimal('73.47'), '60618': Decimal('97.96'), '60622': Decimal('65.31'),
'60623': Decimal('65.31'), '60625': 'one of them is not python file', '60637':
Decimal('95.92'), '60640': Decimal('83.67'), '60643': Decimal('77.55'), '60644':
Decimal('97.96'), '60649': Decimal('57.14'), '60660': Decimal('89.80'), '60667':
Decimal('97.96'), '60673': Decimal('28.57'), '60678': Decimal('81.63'), '60691':
Decimal('87.76'), '60693': 'one of them is not python file', '60697':
Decimal('67.35'), '60699': Decimal('95.92'), '60705': Decimal('75.51'), '60717':
Decimal('97.96'), '60721': Decimal('67.35'), '60723': Decimal('85.71'), '60731':
Decimal('65.31'), '60734': Decimal('71.43'), '60747': Decimal('97.96'), '60758':
Decimal('89.80'), '60761': Decimal('97.96'), '60765': Decimal('97.96'), '60770':
Decimal('59.18'), '60774': Decimal('69.39'), '60781': Decimal('89.80'), '60791':
Decimal('81.63'), '60793': Decimal('71.43'), '60796': Decimal('79.59'), '60797':
Decimal('97.96'), '60810': Decimal('8.16'), '60825': Decimal('89.80'), '60839':
Decimal('97.96'), '8160': Decimal('89.80')}}}
```


其中Decimal所对应的键值是比较样本的所属user_id。

通过对样本答案的难度评估和平均学生作答得分得出此算法题的阈值为89.80。即在相似度评估中，估值大于此阈值的皆将被判为原创性不高的代码。

时空复杂度评分

将此样本代码传入评价接口。可以得出此样本代码的时间复杂度为O(1)，空间复杂度O(1)。

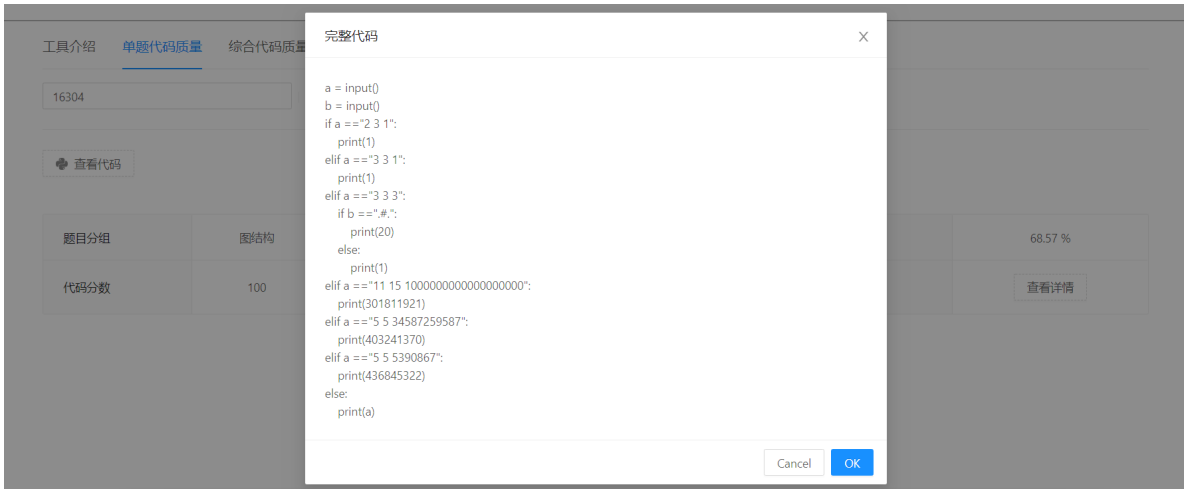
风格评分

将此样本代码传入评价接口。可以得到样本代码风格的17个评价值。

```
is_indent_using_one : True
is_space_nums_multiple_of_four : True
is_within_len_range : True
is_not_trailing_space : True
is_space_around_operator : False
is_not_space_around_operator_in_def : True
is_using_one_quotation : True
is_not_blank_line_beginning : True
is_not_inline_comments : True
is_space_after_pound : True
is_blank_line_after_import : True
is_blank_line_before_class : True
is_blank_line_before_def : True
is_not_diff_package_in_the_same_line : True
is_import_before_from : True
is_not_blank_between_import : True
is_using_meaningful_name : 0%
```

以上三个角度由前端呈现即为：

点击“查看代码”按钮：



可以看到上述案例的评价较为准确。

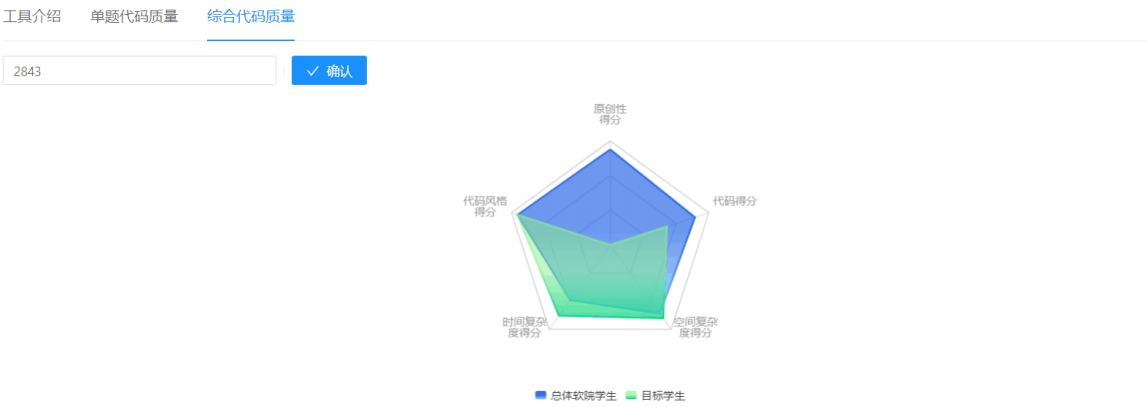
研究问题2

通过对单一学生所有代码评价指标的统计，可以得出学生在样本集中的相对水平高低。

使用**user_id：16304**和**user_id：2843**作为案例分析。

user_id：16304：

user_id：2843：



结题感想

任一丁：

在设计复杂度计算的过程中，经历过多次重构和抽象，反反复复消磨了很多时间。然而，最后进行人工评测的时候依然漏洞百出，靠着反反复复的测试最终才达到了预期效果。总的来说虽然辛苦，但也并非一事无成。

在此非常感谢活跃的队友给予的灵感和帮助，希望今后能将更多的开发工作做得更好。

也感谢陈老师这个学期的教学与帮助，很大的提高了我的能力水平。

林希澄：

最初在设计代码相似度的功能点时，碰到非常多困难。查阅诸多数据后才慢慢有了思路。最复杂的部份还是在于分析 AST 语法树的部分，消磨了许多精力。所幸最后还是圆满的完成这个功能点，准确度也挺好的。

后期我还搭建了网页，为的就是希望可以将项目展示的更加到位。使用 React 作为前端框架，倒是没有遇到太多困难。不过后端因为使用的是 Flask 框架，碰到一些跨域，设计后端接口等等开发上的问题，也花了时间去学习了解，也算是多多增广见识。

对于这个项目还是觉得挺有意义的，也感谢队友们的配合和合作!

谢谢陈老师一学期来的谆谆教诲，对于这学期无法现场听段子备感可惜。谢谢陈老师！

薛人玮：

在设计代码风格的功能点时，没有碰到什么困难。但在处理数据的时候遇到了诸多困难。一是出现了一些超出我们预想的情况，如编程语言不是Python3等。二是在处理统计数据的时候，寻找不到合理的算法去评估各项指标与阈值。三是数据集较为庞大，编写时要时刻注意时空复杂度。在测试过程中，运行时间一直限制了工作的效率。同时也不止一次出现电脑运行内存被挤爆的情况。不过最后在队友的帮助下，也算是成功的完成了项目任务

后期由于对Flask 框架的不熟悉，在实现后端接口是也遇到了问题，也花了时间去学习了解，最后也算是成功完成任务。

在项目过程中，组员们积极沟通，互帮互助，确实大大增加了攻克问题的效率，增加了全组的热情。所以最后一定要感谢小组大家的共同努力，合作愉快！

对于这学期无法现场听陈老师授课备感可惜。陈老师，我想学Python找工作！谢谢陈老师！

附录

附录一：《抽样人工评测表单》

表单详情见代码目录下*record_data/code_complexity_sample.xlsx*

附录二：《人工测评标准》

与机器评测逻辑大致相同，给出时间/空间复杂度判断。

时间复杂度：

1. 逻辑中含有循环关键字的，每一层记为n。
2. 逻辑中含有二分法（多分法）的，每一层记为log_n。
3. 逻辑中含有常见递归的（如斐波那契数列），按照常见递归表单*标记。

空间复杂度：

1. 在循环结构中对列表项进行添加的，每一层记为n
2. 在二分法（多分法）中对列表项进行添加的，每一层记为log_n
3. 利用循环结构对列表进行声明的，每一层记为n
4. 在递归中声明列表并循环添加内容的，每一层记为n

附录三：《常见递归表单》

递归形式	**复杂度**
T(n)=T(n/2)+O(1)	T(n)=O(log_n)
T(n)=T(n-1)+O(1)	T(n)=O(n)
T(n)=2*T(n/2)+O(1)	T(n)=O(n)
T(n)=2*T(n/2)+O(n)	T(n)=O(n*log_n)
T(n)=2*T(n/2)+O(n*log_n)	T(n)=O(n*log^2_n)
T(n)=T(n-1)+O(n)	T(n)=O(n^2)
T(n)=2*T(n-1)+O(1)	T(n)=O(2^n)
T(n)=T(n-1)+T(n-2)+O(1)	T(n)=O(2^n)

PS：其中计算复杂度时，递归式中若包含对同文件其他函数的调用，将其视为O(n）。