

研究报告

1 研究问题

在完成时间相对宽松的情况下，大多数同学都能在OJ编程练习上拿到满分，从而失去区分度。本项目通过考虑多种评价标准，以数据科学大作业的结果数据为样本，进行评分。

应用场景：非限时OJ编程练习

2 研究方法

2.1 数据预处理

我们的分析基于老师给的基础json文件，以及下载的同学们的代码。

在已有的有限的信息的基础上，我们选择分析同学们的提交次数、debug时间、代码的有效行数以及代码的质量。

我们是这样考虑的，编程的能力一般包含debug的能力、代码的质量，以及其他诸如编程工具的了解程度等。但是，我们只能从已有的全体数据中获得这些了。

针对算法题目，我们把oj平台的8个题目分为四种题型：题型一（数字、数组）、题型二（线性表、字符串）、题型三（查找、排序）、题型四（树、图）。

2.1.1 提交次数

我们认为提交次数可以反映同学们的debug水平，提交次数越少则意味着debug能力越强。

于是我们选择同学的提交次数作为一个维度来让进行后续的分析。

2.1.2 debug时间

我们认为，代码的debug时间可以反映同学们的debug水平。代码的首次提交时间和首次得到满分或者最后一次提交的时间的差可以作为代码的提交时间。

但是当我们真的计算出debug时间之后，发现大多数样本的时间在5分钟内，小于30分钟的样本超过总样本的95%。我们意识到，只是本地运行或者在其他ide写代码debug的时候是不会产生提交信息的。等到上传提交时，代码的完成度以及很高了，也经过了不少次debug。这就使得我们的debug能力获得的数据失真。与此同时，我们发现还有极少数样本的debug时间超过了一天，我们将这样的样本过滤掉。

2.1.3 有效行数

我们认为，代码的有效行数（除去注释和空行以外的行数）可以反映代码的简洁性。

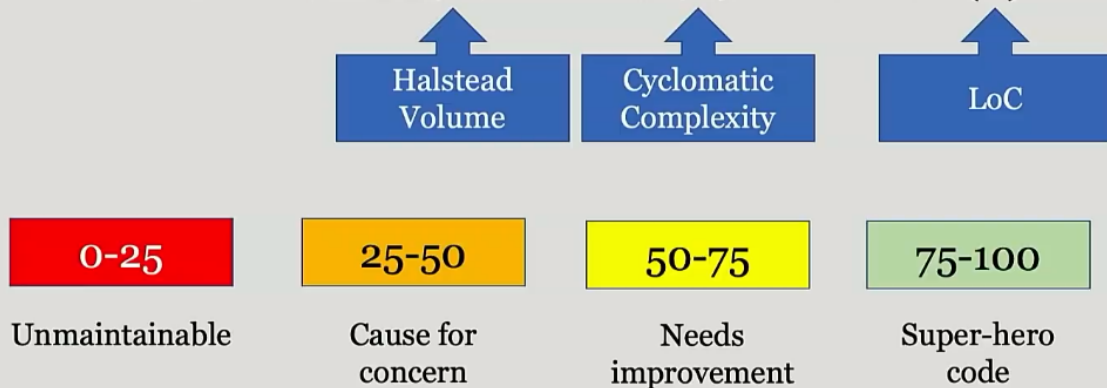
读取有效行数之后，我们惊奇地发现，出现有效行数为0同时最终得分是满分的样本。这样的样本产生的原因不得而知，但是必须被过滤掉。

2.1.4 代码质量

我们使用代码的可维护指数作为代码质量的评判依据。

Maintainability Index

$$MI = 171 - 5.2 \ln(V) - 0.23(C) - 16.2 \ln(L)$$



Cyclomatic complexity是循环复杂度，它是一种软件度量，用于指示程序的复杂度。它是对程序源代码中线性独立路径数量的定量度量。一段源代码的循环复杂度是其中的线性独立路径的数量，其中“线性独立”是指每个路径都具有至少一条不在其他路径之一中的边。例如，如果源代码不包含控制流语句（条件或决策点），则复杂度将为1，因为通过代码只有一条路径。如果代码中有一个单条件IF语句，那么代码中将有两条路径：一条路径IF语句的计算结果为TRUE，另一条路径的IF语句计算结果为FALSE，因此复杂度为2。两个嵌套的单条件IF，或具有两个条件的一个IF，其复杂度为3。

Halstead复杂度度量是Maurice Howard Halstead于1977年提出的软件度量，这是他关于建立软件开发经验科学的论文的一部分。Halstead观察到，软件的度量标准应该反映算法在不同语言中的实现或表达，但与它们在特定平台上的执行无关。因此，这些度量是从代码中静态计算的。计算公式如下图：

For a given problem, Let:

- η_1 = the number of distinct operators
- η_2 = the number of distinct operands
- N_1 = the total number of operators
- N_2 = the total number of operands

From these numbers, several measures can be calculated:

- Program vocabulary: $\eta = \eta_1 + \eta_2$
- Program length: $N = N_1 + N_2$
- Calculated program length: $\hat{N} = \eta_1 \log_2 \eta_1 + \eta_2 \log_2 \eta_2$
- Volume: $V = N \times \log_2 \eta$
- Difficulty : $D = \frac{\eta_1}{2} \times \frac{N_2}{\eta_2}$
- Effort: $E = D \times V$

The difficulty measure is related to the difficulty of the program to write or understand, e.g. when doing [code review](#).

The effort measure translates into actual coding time using the following relation,

- Time required to program: $T = \frac{E}{18}$ seconds

Halstead's delivered bugs (B) is an estimate for the number of errors in the implementation.

- Number of delivered bugs : $B = \frac{E^{\frac{2}{3}}}{3000}$ or, more recently, $B = \frac{V}{3000}$ is accepted^{[[citation needed](#)]}.

在测算代码质量的过程中，我们发现有些代码不是使用Python语言写的。因为我们无法测算非Python语言代码的质量，因此我们将这样的代码过滤掉。

2.2 能力的计算、题型的得分

2.2.1 三种能力的计算

将用户四种题型每种能力对应的得分以1: 1: 1: 1的比例相加得到每种能力的分数。

2.2.2 四种题型的得分计算

用户的分数分为三个部分：debug得分、有效行数得分、mi指数得分。得分的具体计算方式可以参照“dim1分析.md”、“dim2分析.md”、“dim3分析.md”、“dim4分析.md”文档。

2.3 汇总七种能力

在经过数据预处理和对debug能力，代码简洁度，mi指数及四类题型的分析后，我们把这七项数据汇总到了一起，并进行了数据可视化处理，使得最终的结果能给任课老师提供一种直观的视角。

3 代码解释

[项目的GitHub地址](#)

解释代码和项目的关系，代码实现逻辑

3.1 第一阶段代码

3.1.1 读取提交次数

```
for each_id in keys_l:
    the_id = each_id
    to_save_dict[the_id] = {
        'dim1': [],
        'dim2': [],
        'dim3': [],
        'dim4': []
    }
    cases_of_the_id = json_data[the_id]['cases']
    for i in range(len(cases_of_the_id)):
        the_case = cases_of_the_id[i]
```

两重循环进入该题目的数据

```
# 即使没有满分，提交次数和debug时间也是正确的
my_upload_cnt = first_full + 1
```

这里的firstfull是指首次满分的索引

3.1.2 读取debug时间

```

for each_id in keys_l:
    the_id = each_id
    to_save_dict[the_id] = {
        'dim1': [],
        'dim2': [],
        'dim3': [],
        'dim4': []
    }
    cases_of_the_id = json_data[the_id]['cases']
    for i in range(len(cases_of_the_id)):
        the_case = cases_of_the_id[i]

```

两重循环进入该题目的数据

```

# 直接到下一个循环
if len(upload_records_of_the_case) == 0:
    continue
for j in range(len(upload_records_of_the_case)):
    # 找到第一次满分的索引
    if 100 - upload_records_of_the_case[j]['score'] <= 0.1:
        first_full = j
        break

for_name_timestamp = time.localtime(float(upload_records_of_the_case[first_full]['upload_time']) / 1000)
time_name = time.strftime("%Y-%m-%d %H:%M:%S", for_name_timestamp)

# 即使没有满分，提交次数和debug时间也是正确的
my_upload_cnt = first_full + 1
# 单位秒
my_debug_time = (upload_records_of_the_case[first_full]['upload_time'] - upload_records_of_the_case[0][
    'upload_time']) / 1000

```

找到第一次满分的提交，然后依次找到后一个时间，做差

3.1.3 有效行数

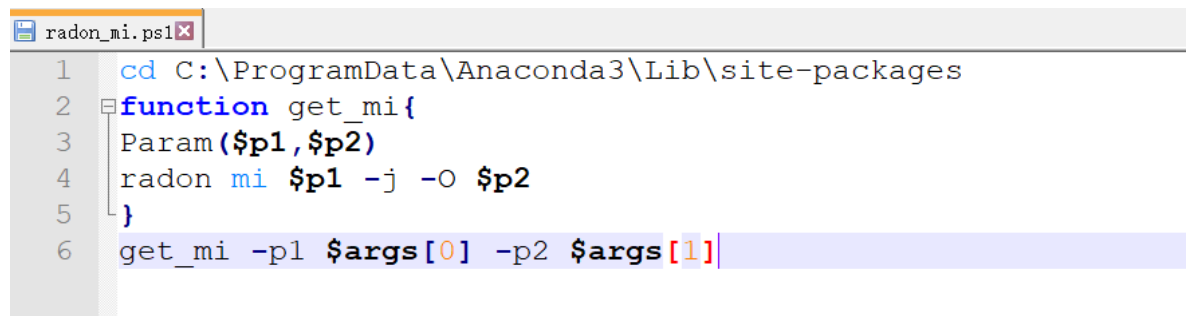
```

with open_file(filename) as f:
    # 按行访问
    for line in f:
        try:
            # 每行添加到列表，返回列表
            line_list.append(line)
        except IOError:
            pass
    count = len(line_list)
    for i in range(len(line_list)):
        content = line_list[i]
        if content == '\n':
            count -= 1
        else:
            for j in range(len(content)):
                if content[j] != " ":
                    if content[j] == "#":
                        count -= 1
                        break
                    else:
                        break

```

先直接得到总行数，然后对每一行的情况做判断。只要，该行是空行或者是注释行就减一，由此得到有效行数。

3.1.4 mi指数



```

1 cd C:\ProgramData\Anaconda3\Lib\site-packages
2 function get_mi{
3     Param($p1,$p2)
4     radon mi $p1 -j -O $p2
5 }
6 get_mi -p1 $args[0] -p2 $args[1]

```

我们使用radon包提供的方法计算Python代码的mi指数

```
def get_mi_json_from_powershell(src_py_f, ans_json_f):
    try:
        args = [r"C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe", "-ExecutionPolicy", "Unrestricted",
                r"E:\新桌面\radon_mi.ps1",
                src_py_f, ans_json_f]
        p = subprocess.Popen(args, stdout=subprocess.PIPE)
        dt = p.stdout.read()
        return dt
    except Exception:
        print("exception")
        return False
```

参数分别是，源代码的地址和生成的表述mi指数的文本文件的地址

```
get_mi_json_from_powershell(tmp_py_file_path, tmp_mi_p)
try:
    with open(tmp_mi_p, 'r') as tmp_mi_json_file:
        tmp_mi_json_data = json.load(tmp_mi_json_file)
        keys_l = list(tmp_mi_json_data.keys())
        my_mi = float(tmp_mi_json_data[keys_l[0]]['mi'])
        tmp_mi_json_file.close()
except KeyError:
    print("user", the_id, "case", c_i, "=====没有mi可能不是Python写的===== ", file=log_f)
    continue
```

在这里，我们将mi值存入字典，即将被存储到json文件中。

3.1.5 整合

四种指数的整合，我们写在了getFours.py文件中。至此，我们得到了indicators_of_four_dim_filtered.json这个文件。

3.2 第二阶段代码

3.2.1 预处理前数据特征绘制



此程序是用于在确定最后的对能力计算的公式选择上作为数据可视化的辅助功能，以及实践使用数据可视化方法。

其中target_dim为目标分析题型，target_num为目标分析题型的数据项，0代表提交次数，1代表debug时间，2代表有效行数，3代表mi指数。第14行代码对目标分析题型数据项进行筛选，主要是便于分析公式选择时候的断点。第20行到第28行代码调用了matplotlib库，对筛选出来的数据进行直方图可视化，以便我们在数据分析时对数据有直观认识。

3.2.2 可视化分析（以dim2为例）

```

#
5   target_dim="dim2" #目标分析类型为dim2
6
7   x1=[] #提交次数
8   x21=[] #debug时间(一分钟以上)(数据预处理中舍去了超过24小时的样本)
9   x22=[] #debug时间(三十分钟以上)
10  x3=[] #代码有效行数
11  x4=[] #mi指数
12  count=0
13  count_oneMinute=0
14  count_30Minute=0
15  count_1submit=0
16  count_50line=0
17  count_100mi=0
18  with open("../.../indicators_of_four_dim_filtered.json",'r') as fp:
19      df=json.load(fp)
20      for item in df:
21          if(len(df[item][target_dim])>0):
22              for sub_item in df[item][target_dim]:
23                  count+=1
24                  if(sub_item[0]==1):
25                      count_1submit+=1
26                  if(sub_item[2]<=50):
27                      count_50line+=1
28                  if(sub_item[3]>100):
29                      count_100mi+=1
30                  x1.append(sub_item[0])
31                  x3.append(sub_item[2])
32                  x4.append(sub_item[3])
33                  if(sub_item[1]>60):
34                      x21.append(sub_item[1])
35                      count_oneMinute+=1
36                  if(sub_item[1]>60*30):
37                      x22.append(sub_item[1])
38                      count_30Minute+=1

```

代码第18行使用的json文件，是在第一阶段的数据预处理后得到的经过一些条件筛选过滤的数据集，x1-x4数组记录了此数据集中的dim2的数据项。

```

num_bins=[50,500,50,50]
plt.hist(x1,num_bins[0])
plt.xlabel("提交次数")
plt.ylabel("频数")
plt.title(r"提交次数分布情况")
plt.show()

plt.hist(x21,num_bins[1])
plt.xlabel("debug时间(超过1分钟)")
plt.ylabel("频数")
plt.title(r"debug时间分布情况(超过一分钟)")
plt.show()

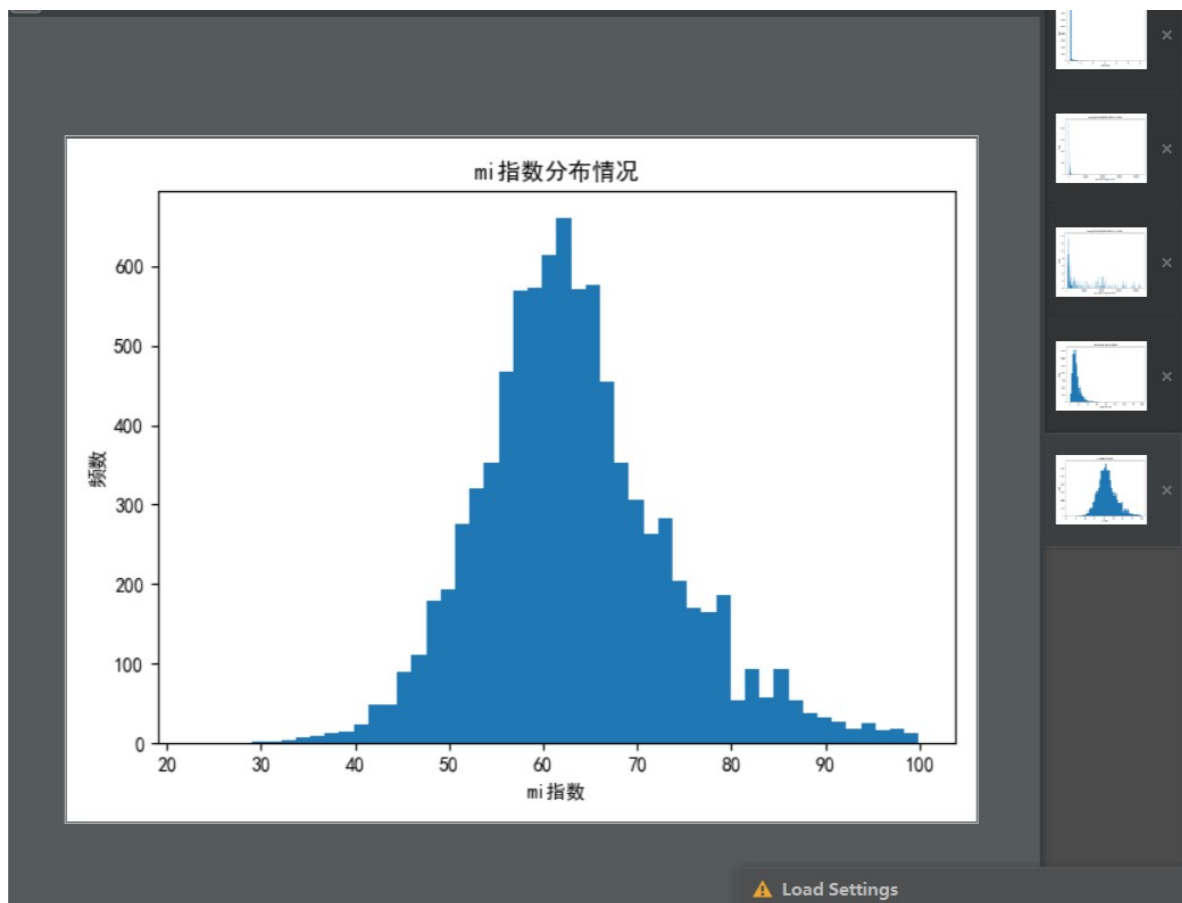
plt.hist(x22,num_bins[1])
plt.xlabel("debug时间(超过30分钟)")
plt.ylabel("频数")
plt.title(r"debug时间分布情况(超过三十分钟)")
plt.show()

plt.hist(x3,num_bins[2])
plt.xlabel("有效代码行数")
plt.ylabel("频数")
plt.title(r"有效代码行数分布情况")
plt.show()

plt.hist(x4,num_bins[3])
plt.xlabel("mi指数")
plt.ylabel("频数")
plt.title(r"mi指数分布情况")
plt.show()

```

num_bins数组中存储的调用matplotlib库画图时使用的区间大小参数。接下来的5个代码块实现了将dim2中数据项可视化的功能。运行此方法得到的结果图如下。



3.2.3 计算在dim2上用户的综合得分情况

```

34 target_dim="dim2"
35 count=0
36 time_list=[]
37 line_list=[]
38 mi_list=[]
39 sumLine=0
40 sumMI=0
41 with open(".././././indicators_of_four_dim_filtered.json", 'r') as fp:
42     df=json.load(fp)
43     for item in df:
44         if(len(df[item][target_dim])>0):
45             time_list.append(calDebugTimeScore(df[item][target_dim]))
46             for subItem in df[item][target_dim]:
47                 count+=1
48                 sumLine+=subItem[2]
49                 sumMI+=subItem[3]
50     avgLine=sumLine/count
51     avgMI=sumMI/count
52     for item in df:
53         if(len(df[item][target_dim])>0):
54             line_list.append(calLineScore(df[item][target_dim], avgLine))
55             mi_list.append(calMIScore(df[item][target_dim], avgMI))
56     score_list=[]
57     for i in range(0, len(time_list)):
58         score_list.append(0.5*time_list[i]+line_list[i]+1.5*mi_list[i])
59     plt.hist(score_list, 50)
60     plt.xlabel("得分")
61     plt.ylabel("频数")
62     plt.show()

```

第43行至第51行间，计算了整个样本中所有数据的平均有效行数和平均MI指数，用于下一步计算单个用户的得分，同时第45行计算了单个用户的debug时间得分，第54行、55行计算了单个用户的有效行数得分和mi指数得分。计算得分的具体方法参见下图。


```

2 import matplotlib.pyplot as plt
3 plt.rcParams['font.sans-serif']=['SimHei']=显示中文
4
5 def calDebugTimeScore(list):
6     sumScore=0
7     n=0
8     for subItem in list:
9         n+=1
10        if(subItem[1]<30*60):
11            sumScore+=1
12        elif(subItem[1]>=30*60 and subItem[1]<130*60):
13            sumScore+=1-((subItem[1]-30*60)/60)*0.01
14        else:
15            sumScore+=0
16    return sumScore/n
17
18 def callineScore(list, avgLine):
19     n=0
20     sumScore=0
21     for subItem in list:
22         n+=1
23         sumScore+=avgLine/subItem[2]
24     return sumScore/n
25
26 def calMIScore(list, avgMi):
27     n=0
28     sumScore=0
29     for subItem in list:
30         n+=1
31         sumScore+=(avgMi/subItem[3])**2
32     return sumScore/n

```

3.2.4 按单项能力对用户评分

```

with open(".././../indicators_of_four_dim_filtered.json", 'r') as fp:
    df=json.load(fp)
    for item in df:
        if(len(df[item]["dim1"])):
            user_ids.append(item)
            for target_dim in target_dims:
                if (len(df[item][target_dim]) > 0):
                    if(target_dim=="dim1"):
                        time_list_dim1.append(calDebugTimeScore(df[item][target_dim]))
                        for subItem in df[item][target_dim]:
                            count += 1
                            sumLine += subItem[2]
                            sumMI += subItem[3]
                        avg_lines.append(sumLine/count)
                        avg_mis.append(sumMI/count)
                        sumLine = 0
                        sumMI = 0
                        count = 0

```

先计算每个题型，所有用户的平均有效行数和平均MI指数，并同时计算该用户的debug时间得分。图上所见为计算题型一（数字、数组）的方法，计算其他题型的方法类似。

```

for item in df:
    for target_dim in target_dims:
        if (len(df[item][target_dim]) > 0):
            if(target_dim=="dim1"):
                line_list_dim1.append(calLineScore(df[item][target_dim], avg_lines[0]))
                mi_list_dim1.append(calMIScore(df[item][target_dim], avg_mis[0]))
            elif(target_dim=="dim2"):
                line_list_dim2.append(calLineScore(df[item][target_dim], avg_lines[1]))
                mi_list_dim2.append(calMIScore(df[item][target_dim], avg_mis[1]))
            elif(target_dim=="dim3"):
                line_list_dim3.append(calLineScore(df[item][target_dim], avg_lines[2]))
                mi_list_dim3.append(calMIScore(df[item][target_dim], avg_mis[2]))
            else:
                line_list_dim4.append(calLineScore(df[item][target_dim], avg_lines[3]))
                mi_list_dim4.append(calMIScore(df[item][target_dim], avg_mis[3]))

```

然后再使用每种题型的平均有效行数和平均mi指数作为参数计算每个用户在该题型上的得分情况。

```

126 for i in range(0, 207):
127     user_time_list.append(time_list_dim1[i]+time_list_dim2[i]+time_list_dim3[i]+time_list_dim4[i])
128     user_line_list.append(line_list_dim1[i]+line_list_dim2[i]+line_list_dim3[i]+line_list_dim4[i])
129     user_mi_list.append(mi_list_dim1[i]+mi_list_dim2[i]+mi_list_dim3[i]+mi_list_dim4[i])
130
131     to_save_dict = {}
132     for each_id in user_ids:
133         to_save_dict[each_id] = {
134             'debug_time':0.0,
135             'code_line':0.0,
136             'mi_score':0.0
137         }
138     i=0
139     for user_id in user_ids:
140         to_save_dict[user_id]['debug_time']=user_time_list[i]
141         to_save_dict[user_id]['code_line']=user_line_list[i]
142         to_save_dict[user_id]['mi_score']=user_mi_list[i]
143         i+=1
144     with open(outputs_path, 'w+') as r:
145         # 定义为写模式，名称定义为r
146         json.dump(to_save_dict, r)
147         # 将dict写入名称为r的文件中
148         r.close()

```

最后再综合每个用户四种题型的得分，再记debug时间得分、有效行数得分和mi指数得分写入到一个json文件中。

3.2.5 汇总雷达图的数据

通过代码“pre-processed_data_to_7areas_score.py”将四种题型的分数和用户的三种能力的分数汇总起来存储在“7areas_score.json”中，为后续雷达图的绘制提供数据。

3.2.6 绘制雷达图

```

src_path = r'../json_flies/7areas_score.json'
plt.rcParams['font.sans-serif'] = ['SimHei'] # 显示中文
with open(src_path, "r") as fp:
    # 预处理的数据
    src_data = json.load(fp)
    fp.close()
    # 所有人的id
    user_ids = list(src_data.keys())
    for each_id in user_ids:
        plt.clf()
        df = pd.DataFrame({
            'group': ['userA'],
            '题型四': [src_data[each_id]['case4']],
            'mi指数': [src_data[each_id]['mi_score']],
            '代码简洁性': [src_data[each_id]['v1_score']],
            'debug能力': [src_data[each_id]['dt_score']],
            '题型一': [src_data[each_id]['case1']],
            '题型二': [src_data[each_id]['case2']],
            '题型三': [src_data[each_id]['case3']]
        })

        out_path = '../pngs/radar_chart' + '_user_' + each_id + '.png'

```

绘制四种题型得分和三种能力得分的雷达图并存储

4 案例分析

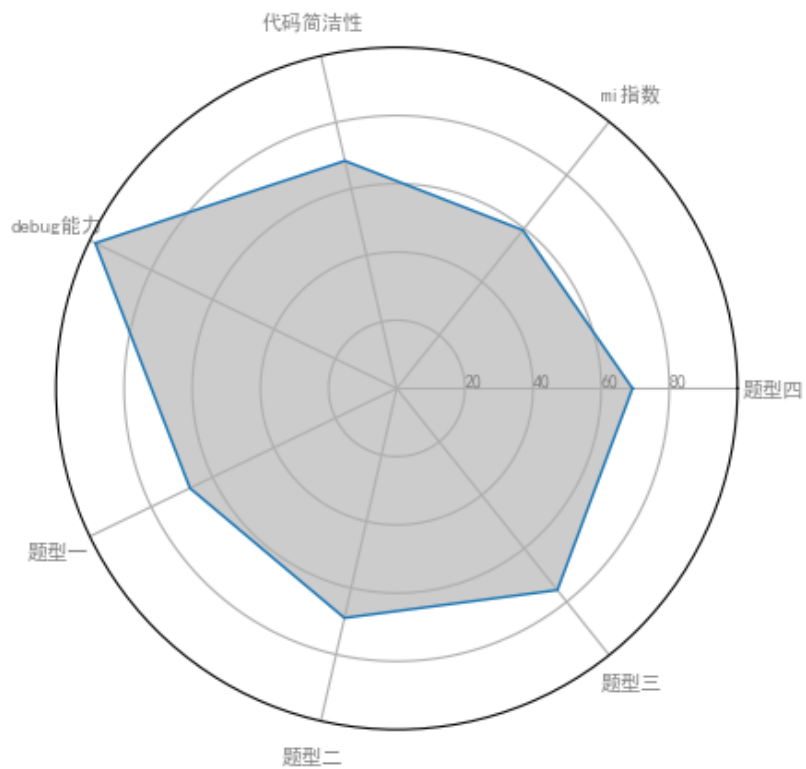
以user_id为48117的样本为例。

```

"48117": {
  "dim1": [
    [
      1,
      0.0,
      10,
      64.01604595547325
    ],
    [
      8,
      10835.022,
      27,
      49.90485240214782
    ],
    [
      7,
      1138.678,
      27,
      53.9574242203185
    ],
    [
      8,
      19728.445,
      25,
      52.11258738116969
    ],
    [
      2,
      12.905,
      18,
      59.6668854374987
    ],
    [
      3,
      46.071
    ]
  ]
}

```

上图为user_id为48117的用户的编程练习结果在经过第一阶段数据预处理后的部分数据指标，其中"dim1"指代题型一（数字，数组）。



上图为经过7种指标的计算和对结果数据的可视化处理后后，48117用户的雷达图。

5 对课程的意见

6 附录

参考资料

[radon计算工具](#)

[循环复杂度](#)

[Halstead复杂度度量](#)

[代码质量评估](#)