数据科学基础大作业报告

目录

一、小组信息：

✧ 小组人数：3

✧ 组员名单：

姓名	学号	邮箱	python 完成习题数
张嘉玥	181250185	181250185@smail.nju.edu.cn	200
吴雨晴	181250161	181250161@smail.nju.edu.cn	189
冯泊涓	181250030	181250030@smail.nju.edu.cn	198

✧ 组员分工职责：（序号表示时间排序）

张嘉玥	3.题型均分、个人均分 5.个人能力雷达图绘制 8.代码完成量变化趋势图相似度匹配-动态时间规整（DTW）算法 8.个人能力雷达图匹配 9.代码完成量变化趋势图相似度匹配-对比图绘制 10.代码整合 11、ppt 制作+部分报告撰写
吴雨晴	2.代码分析净化，题目均分、平均提交次数、平均代码行数 5.每日代码完成量整理 6.代码完成量变化趋势图绘制 8.代码进度趋势相似度匹配-求导差算法 9.代码完成量变化趋势图相似度匹配-对比图绘制 10.代码整合 11、视频制作+部分报告撰写
冯泊涓	1.分组题目及名单整理 4.题目难度系数分析 7.代码完成量变化趋势图非线性回归分析 8.代码进度趋势相似度匹配-余弦相似度算法 9.代码完成量变化趋势图相似度匹配-对比图绘制 10.代码整合 11、报告撰写

二、研究问题

1、研究问题：

以代码时间习惯为主，代码多维能力为辅——多模式“拖延症克星”编程 cp 匹配

2、代码开源地址：https://github.com/BigCode-Goood/master

三、研究方法

1、研究思路：

a、创意背景：

随着线上教学体系不断发展完善、投入使用，其高效、便捷等种种优势显然是传统线下教学模式所无法比拟的。但是线上教学有其与生俱来的痛点——对于**学生自觉性**的高度依赖。这是线上教学体系优势发挥的巨大掣肘。

今年年初，在疫情爆发的不可抗力影响下，线上教学的发展被猛烈地往前推了一把。和在校学习生活不同，突如其来的网络教学、单机编程让我们彼此失去了周身环境的“参考系”，失去了同学的日常“鞭策”。“自主学习能力”几乎是完全决定了一个人线上学习的质量。在这种情况下，有人能将自己学习生活时间安排得“游刃有余”，也有人受困于“拖延症”，成为了 DDL 驱动型学习者。

回想起奋斗于 200 道 python 题的那一个多月，我们一致认为，在学生能力和题目难度大致匹配的情况下，决定学生任务完成度的首先是时间安排的合理性，其次才是学生编程水平的高低。结合研究题目中“编程 cp”一项，我们决定将时间安排与 cp 匹配结合起来，以时间安排为主，代码能力为辅，进行**“拖延症克星”编程 cp 匹配**，以**提高学生学习的自觉性和时间安排能力**。

b、研究思路：

确定研究内容为“拖延症克星”编程 cp 匹配后，我们探讨回答了以下几个问题，研究思路逐渐清晰。

1、如何反映学生代码时间安排习惯?

自然而然地，我们想到了画每日任务完成量折线图的方式。通过观察任务完成量-时间折线图的走向趋势，我们可以看出一个学生是习惯“前期多做些，后面少做点”，还是“任务均摊到每天”，还是“DDL 驱动”。每日任务完成量可以通过这一天完成题目总分得到。

但是事实上，由于题目之间难度差异较大，单纯的分数并不能完全反应学生的精力投入。为此，我们决定通过“题目难度系数”对题目难度带来的精力投入误差进行修正。

同时，考虑到以“每日”作为时间单位可能导致折线图产生巨大的波动，不能很好地展现代码时间习惯，我们决定在后续作图时尝试不同天数的时间单位，对比择优。

2、如何反应学生代码能力?

经过题目难度系数修正后，学生的总分已经可以代表其代码能力水平。考虑到题目类型的多元性，我们决定分别计算学生在每一类型题目的能力，并采用雷达图来展现。

是否要直接采用修正总分衡量能力呢？考虑到修正系数的不可控性，修正后的分数事实上失去了“100 分满分”的参考，分数本身没有很大的意义。经过取平均、利用正态分布分析总体等方法的提出和否决，最后我们决定直接采用最简单直接粗暴的办法——根据分数排名取百分比。

3、如何衡量题目难度?

我们初步考虑的指标包括：题目类型均分、题目平均分、提交次数、做题时间、代码行数。但是后来我们发现基于现有数据，精确的做题时间很难求得，不仅不能提高准确性，反而引入了极大的误差，所以我们舍弃了这一指标。

最后参考的指标为：题目类型均分、题目平均分、提交次数、代码行数，共 4 个指标。

考虑每一指标的数量级和重要程度，我们选择对均分取倒数、对提交次数和代码行数取对数。在得出题目难度系数图像后，再看情况对具体公式做出调整。

为了得到每一道题目的平均分、提交次数、代码行数等等信息，难免需要对所有题目提交记录遍历分析。如果把每一道题目的每一份提交记录都分析一遍，那就需要分析 40000 条提交记录，我们认为这是一件很低效的事情。结合助教阐述的分组题目情况，每组间的题目交叉并不多。故而五组题目的做题记录事实上是五份独立性极高的样本，我们可以通过学生做题记录的交叉还原出分组情况，只针对其中一组进行分析。

4、如何分析学生代码时间安排习惯的相似度?

我们已经确定使用任务完成量变化趋势折线图来反映代码时间习惯。如何对折线图进行相似度分析？首先，我们认为做题量的变化趋势是有大致的分类的，例如上升、下降等。所以我们选择先对折线图进行曲线拟合，以便于对折线图进行趋势分类。再按照分类对图像进行相似度分析。

5、如何利用时间安排习惯和能力给学生匹配 cp?

起初，我们打算将时间安排习惯相似的学生匹配为编程 cp，以达到“赛跑”一般的竞争效果，互相激励。同时，可以结合学生能力的相似或者互补进行匹配。考虑到编程 cp 难以双向选择，可能存在 A 的最佳匹配是 B，但 B 的最佳匹配是 C 的情况。为了避免“凑对”涉及到的衡量取舍，我们决定直接考虑“单箭头”的匹配模式。

在后期观察匹配结果图像时，我们发现：大部分匹配图像趋势十分相似，匹配效果很好，但是也存在个别匹配效果较差的 cp。 由此，我们又产生了进行“互补 cp 匹配”的想法。 比起相似 cp“同进退”的督促方式，互补 cp 也可以起到“我在休息你在学习”的互补督促效果。

经过讨论，考虑到各项任务的数据需要，我们将问题初步分为以下 5 个步骤，再在各个阶段通过会议细化阶段性任务。

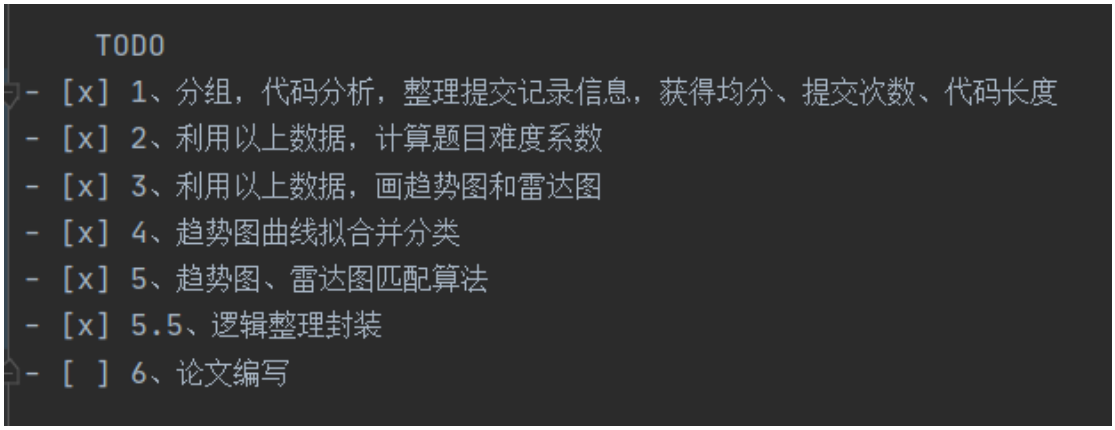

图 1 第一次会议：待办事项清单

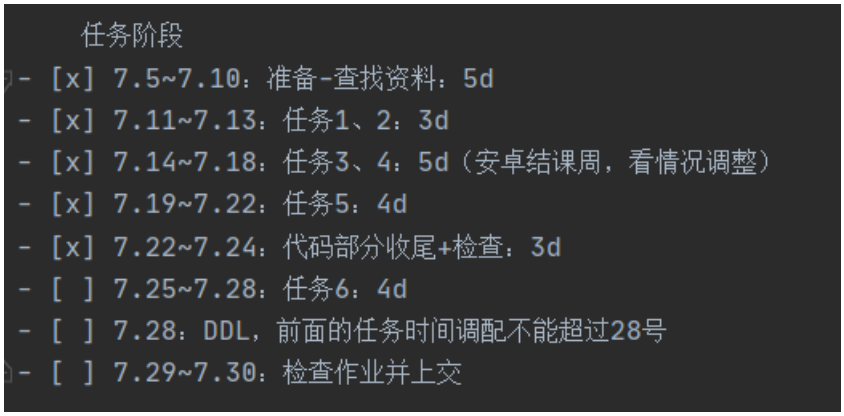

图 2 第一次会议：任务时间安排

c、预期成果：

本研究成果将适用于：为一组有多次中长期刷题任务的学生进行编程 cp 配对。换言之，在进行配对之前，需要至少一次中长期的做题提交记录，以供数据分析。

我们将以绘制代码进度趋势图为主、代码能力雷达图为辅（后称趋势图、雷达图），对学生代码习惯和能力进行相似、互补两种模式的匹配。

✧ 匹配流程：

1、学生首先选择趋势图匹配模式：相似/互补，并输入预期 cp 能力水平

2、根据三种趋势图匹配方式（动态时间规整算法匹配，余弦相似度匹配，求导差匹配）匹配多个候选 cp，筛选出能力水平符合要求的 cp，最后取并集获得候选 cp 列表

3、为学生和所有候选 cp 进行代码能力雷达图匹配，标记雷达图匹配模式：相似/互补/各有所长

4、向学生展示候选 cp 列表，学生可以根据雷达图匹配结果选择 1 人作为编程 cp（匿名）

5、在后续做题任务中，该学生可以获知该编程 cp 的编程动态，与之匿名交流。

注：*编程 cp 非双选制，如 A 选 B，B 选 C，DEF 选 A

***DTW 算法匹配不能很好地匹配趋势图互补类型的 cp，故匹配模式为互补时不考虑 DTW**

✧ 预期匹配效果：

1、趋势图相似匹配，可以达到“同进退”的“竞争”效果，激发积极性。

2、趋势图互补匹配，可以达到“我在休息，你在学习，让我不好意思休息”的效果，互相追赶。

3、能力图相似匹配，可以达到“共同探讨”的效果。

4、能力图互补匹配，可以达到“互相请教”的效果。

2、研究过程与细节处理：

第一阶段：分组，代码分析整理提交记录信息，获得均分、提交次数、代码长度

在进行代码分析和提交记录整理时，我们只保留最后一次提交的信息。考虑到做题过程中存在大量非 python 代码提交和面向用例的情况，我们对非法提交进行了过滤，将其分数和代码行数清零。

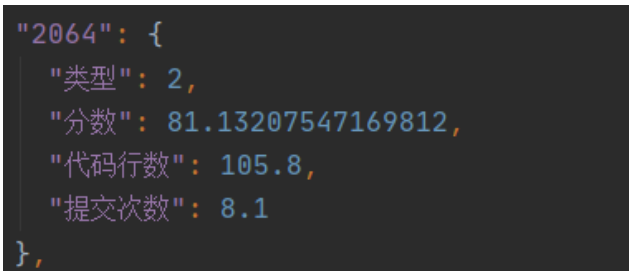

图 3 题目信息整理

图 4 题目类型取值对应关系

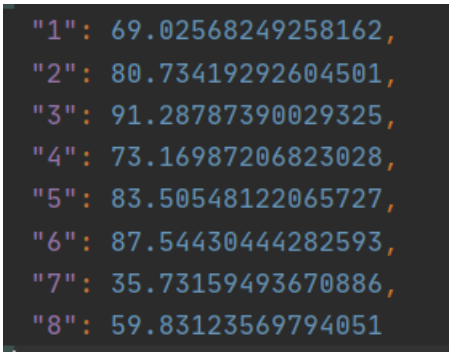

图 5 题目类型均分

第二阶段：计算题目难度系数

考虑每一指标的数量级和重要程度，我们选择对均分取倒数、对提交次数和代码行数取对数。在获取平均提交次数和平均代码行数时，我们对异常提交（非 python 代码、面向用例等）进行了剔除，只计算有效提交记录的平均提交次数和平均代码行数。

令 dif 为题目难度系数，case_avg 为题目均分，type_avg 为题型均分，code_length 为题目平均代码行数，uploads 为题目平均代码行数，初步得到的公

式为：

$$Dif=（100/case\_avg）*（100/type\_avg）*(1+\log_{100} code\_length)*(1+\log_{10} uploads)$$

Dif：题目难度系数　　Case_avg：题目均分　　Type_avg：难度均分
Code_length：代码行数　　Uploads：提交次数

初步运行并绘制折线图后，我们发现存在整体题目难度系数偏高、困难题的难度系数过高甚至于断层的情况。打印部分题目难度系数因子后，我们意识到关键问题在于部分题目均分过低、部分类型均分过低，导致其倒数系数过大。为了让题目难度系数处于 1~3 的合理区间，我们逐渐为不同的系数因子加上不同的系数和次数，并限制了难度系数值区间。

最终采用的公式为：

$$Dif = [（100/case\_avg）^{0.8} *（100/type\_avg）^{0.7} * (1+0.3*\log_{150} code\_length) * (1+0.2*\log_{10} uploads)] * 0.05 + 0.92$$

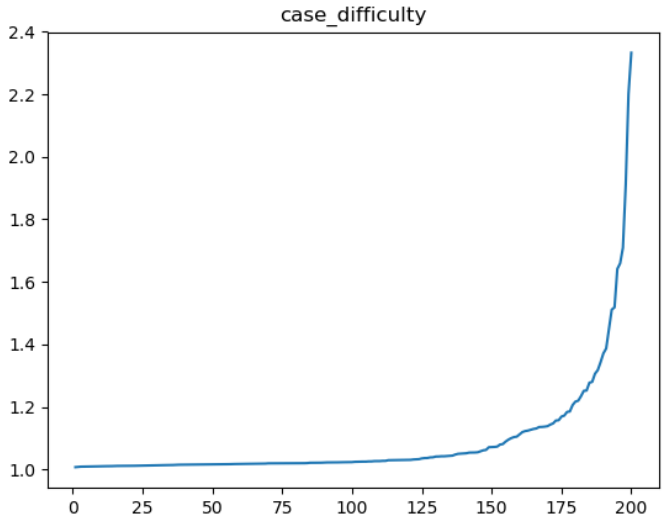

图 6 题目难度系数折线图

由图可见，最终得到的难度系数折线图中，简单题（1.0~1.1）大约为 150 题，中难题（1.1~1.4）大约为 30 题，难题（1.4~2.4）大约为 20 题，基本符合预期。

第三阶段：画趋势图和雷达图

✧ 趋势图

在已分析计算出题目难度的基础上，我们结合每位学生每日做题数量、对应题目得分以及题目难度加权，通过公式“当日完成度=Σ（题目*题目难度系数）”计算出当日完成度。最终将每位学生完成日期与当日完成度作为分析结果成对存为 json 文件以便后续取用。

获取完整的个人完成度数据后，我们将该数据可视化，画出学生在整个作业期间的题目完成趋势图，横轴为日期，纵轴为当日完成度。

在后续的对学生完成趋势图做非线性拟合时，我们发现现有数据存在十分严重的问题：若学生 A 在 3 月 1 日和 3 月 3 日都有做题记录，但 3 月 2 日没有做题，那么画趋势图的时候会直接忽略 3 月 2 日，导致横轴的时间单位划分不均匀，影响趋势图的准确性。

我们需要对数据进行优化。于是我们添加了为完成度“补 0 占位”的逻辑，保证在最后一次提交日期之前，每一日都有完成度的记录——不论完成度是否为 0。这样一来，我们可以更加诚实地展示学生每日做题量的变化趋势，为后续函数拟合提供更完整的数据，提高我们对学生编程习惯分析的科学性。

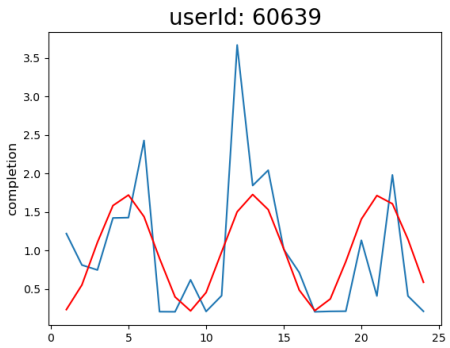

图 7-1 补 0 前拟合图

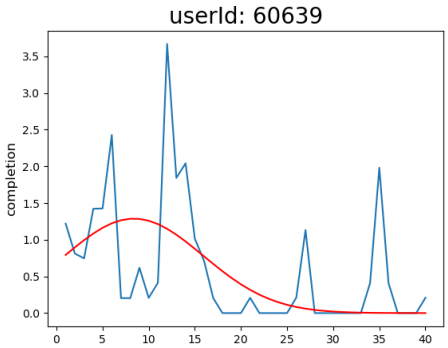

图 7-2 补 0 后拟合图

✧ 雷达图

我们对于雷达图各维指标的计算较为简单，直接通过学生该类型题目完成度在总人数中的排位来衡量学生的能力。如此可以直观地反应个人在整体中的能力处于什么位置。为了符合雷达图面积大小与能力强弱的直观联系，我们将雷达图每一维度范围设为 0 到 1，取值为学生该类题目得分在总体中超越学生人数的百分比。

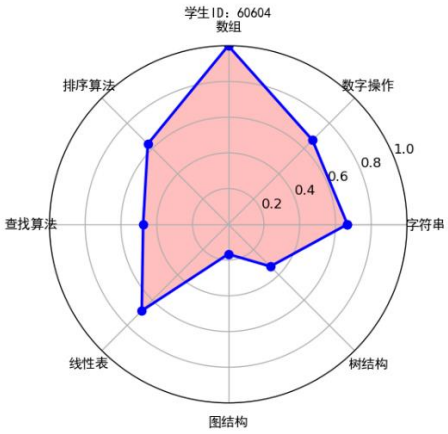

图 8 学生能力雷达图示范

第四阶段：趋势图曲线拟合并分类

为了体现学生习惯的做题趋势，获得一个较为泛化的模型，我们决定给做题趋势折线图做非线性回归分析，获得其相关系数最高的拟合函数，进而对函数进行分类。

我们起初在备选的函数模型中添加了三次等次数较高的多项式函数，本意为拟合较为复杂的趋势变化。但是事实上因为高次多项式函数图像的图像十分复杂，它们往往能够很好地拟合出做题趋势折线中微小的变化，压倒性的复杂度优势导致许多具有典型抛物线特征、指数函数特征的趋势图都被拟合成了高次多项式函数。

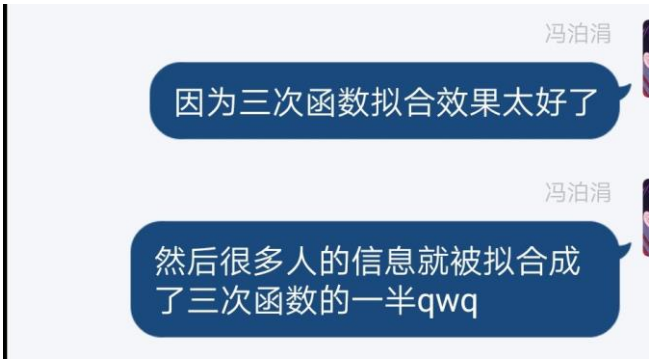

图 9 讨论过程

最后我们选择抛弃高次多项式、S 型函数、幂函数、对数函数等拟合效果过好或过差的函数模型，选择了高斯函数、指数函数、正弦函数、反比例函数、一次多项式、二次多项式等 6 个函数模型。

我们使用最小二乘法进行函数拟合，编写了不同模型的拟合函数，利用表驱动的模式进行遍历调用拟合，选取相关系数 R 最大的一种函数拟合方式。

尝试拟合的过程中，我们又发现了一些拟合系数带来的问题。如高斯函数、二次函数的极值不能在自变量取值范围内取到；指数函数、一次函数上升与下降趋势的不同；正弦函数在自变量取值范围内的周期数极大影响趋势判断……

为了更好地体现趋势，我们细化了函数拟合的分类和限制：

```
if type == 0:  # 高斯函数期望要求
    if popt[1] <= len(x)*0.25 or popt[1] >= len(x)*0.8:
        type += 1
        continue
```

图 10-1 高斯函数峰值位置在 x 轴取值范围的 25%到 80%

```
if n == 2:
    a = res["系数"][0]
    b = res["系数"][1]
    T = -b / (2 * a)  # 求出拐点，若拐点不在图像显示区间内则抛弃
    if T <= 2 or len(x) * 0.8 <= T:
        type += 2
        continue
```

图 10-2 二次函数峰值位置在 x 轴取值范围内

```
if type == 1:  # sin周期要求
    T = np.abs(2 * np.pi / popt[1])
    if len(x) <= 2 * T:
        type += 1
        continue
```

图 10-3 正弦函数在图像显示范围内至少需展示 2 个周期，展示波动性

```
if type == 2:  # 指数函数趋势判断
    if popt[1] < 1: fit_type += 1
```

图 10-4 指数函数判断其上升或下降趋势

至此，我们的拟合类型表为：

```
# 用于拟合的函数
func_list = [func1,func5, func6, func7]
type_list = ["高斯函数", "正弦函数", "上升指数函数", "下降指数函数", "反比例函数",
             "一次增函数", "一次减函数", "U型二次函数","n型二次函数"]
```

图 11 表驱动代码片段

拟合函数与趋势类型的对应字典为：

```
trend_type= {
    "高斯函数": "上升-下降",
    "正弦函数": "波动",
    "上升指数函数": "上升",
    "下降指数函数": "下降",
    "反比例函数": "下降",
    "一次增函数": "上升",
    "一次减函数": "下降",
    "U型二次函数": "下降-上升",
    "n型二次函数": "上升-下降"
}
```

图 12 拟合函数与趋势类型对应字典

考虑到不同的时间单位大小可能会导致不同的拟合效果，我们对时间单位的选择进行了讨论：是 1 天，2 天，还是 3 天？我们认为，对趋势图进行曲线

拟合最重要的成果在于“趋势分类”，而趋势分类本身是一个粗略、直观的行为。“趋势分类”最大的贡献是，在需要匹配的学生数据较多时，按照趋势分类给学生初步划定匹配的范围，可以大幅度提高效率。所以，根据 1 天或是 3 天的折线图进行分类，并不会对后续的相似度分析产生较大的影响。我们选择直接遍历三种时间单位长度，给每一个学生选择相关系数 R 最高的一种。

```
for set_period_length in [1, 2, 3]:  # 设置几天为一个period
```

图 13 遍历时间单位长度

考虑到部分折线图的拟合效果并不好（R<0.65），我们筛选了这一部分趋势图，将其和正弦函数拟合一起标记分类为“波动”

最终我们生成了记录有拟合函数类型、拟合函数系数、拟合相关系数、趋势类型的 json 文件，并绘制了本组每一位同学的函数拟合图，存为图包。


```
"60581": {
  "拟合类型": 3,
  "拟合函数": "下降指数函数",
  "趋势": "下降",
  "相关系数R": 0.8023321403502734,
  "coefficient": [
    25.544904884846684,
    0.9509351090043126,
    -16.170519056614648
  ],
```


图 14 趋势图拟合信息

第五阶段：趋势图、雷达图匹配算法

✧ 趋势图匹配算法

经过讨论，我们选择了 3 种曲线相似度分析方法进行匹配。

A、余弦相似度匹配

余弦相似度通过计算两条曲线中，对应两向量之间夹角余弦值之和来确定相似度。我们选择直接使用折线图，而不是拟合曲线来进行分析，是因为余弦相似度匹配较为精细，是针对这一次做题过程进行的分析和匹配。

```python
def cosine_similarity(x, y, norm=False):
    """ 计算两个向量x和y的余弦相似度 """
    assert len(x) == len(y), "len(x) != len(y)"
    zero_list = [0] * len(x)
    if x == zero_list or y == zero_list:
        return float(1) if x == y else float(0)

    res = np.array([[x[i] * y[i], x[i] * x[i], y[i] * y[i]] for i in range(len(x))])
    cos = sum(res[:, 0]) / (np.sqrt(sum(res[:, 1])) * np.sqrt(sum(res[:, 2])))

    ...

    return 0.5 * cos + 0.5 if norm else cos  # 归一化到[0, 1]区间内
```

图 15 余弦相似度算法

根据匹配结果生成的对比图看来，大部分匹配图像趋势十分相似，匹配效果很好，但是也存在个别匹配效果较差的 cp（余弦相似度 0.5~0.6/1）。由此，我们又产生了进行“互补 cp 匹配”的想法。在相似 cp 的相似度并不高的情况下，可以选择做题趋势图差别最大的 cp 匹配方式，且根据匹配结果，最互补 cp 的趋势相似度基本足够低。比起相似 cp“同进退”的督促方式，互补 cp 也可以起到“我在休息你在学习”的互补督促效果。最终我们匹配得到两组 cp。

结合每位同学的做题趋势类型，我们一开始采用了根据趋势分类匹配的方式。相似 cp 在同类型内匹配，互补 cp 在相反类型内匹配（上升-下降，上下上-下上下）。对于匹配效果不够好的同学，我们再将它们放入“波动”趋势一起匹配。这样在样本容量大的情况下可以极大节省时间和空间。

后续考虑到在样本容量小的情况下，我们本次选择了全部遍历匹配并保存数据的方法，以确保可以取到足够的 cp 供后续匹配进行选择。

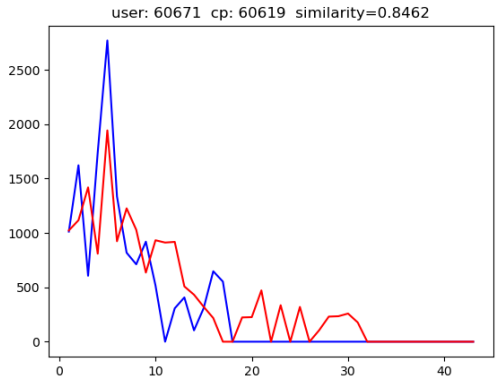

图 16-1 相似 cp 匹配：60671-60619

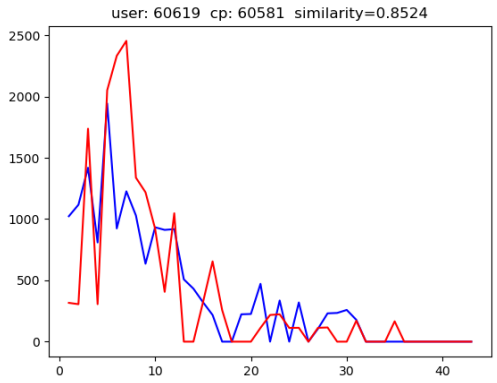

图 16-2 相似 cp 匹配：60619-60581

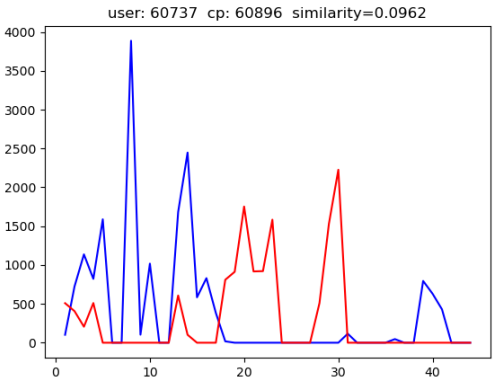

图 16-3 互补 cp 匹配：60737-60896

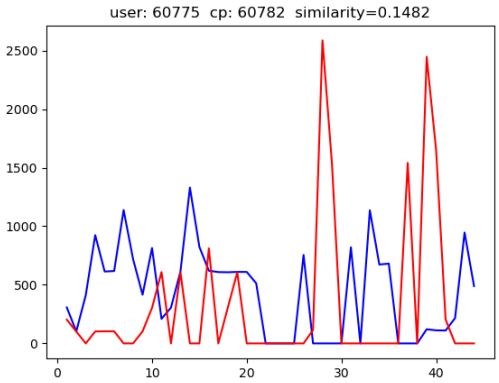

图 16-4 互补 cp 匹配：60775-60782

B、动态时间规整算法（DTW）匹配

DTW（Dynamic Time Warping ）动态时间规整算法，它的出现解决了**传统欧氏距离算法**在计算**“非等长时间序列相似度”**时，因**“对时间上动态变化的忽视”**造成相似度判断的巨大误差的问题。它是一种衡量两个长度不同的时列的相似度的方法，是一个对于点序列取最短路径和的算法。

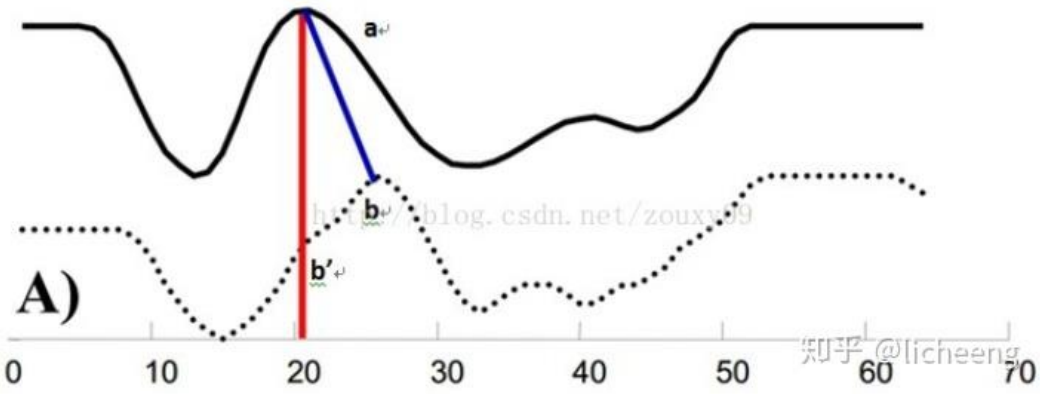

图 17-1 传统欧式算法

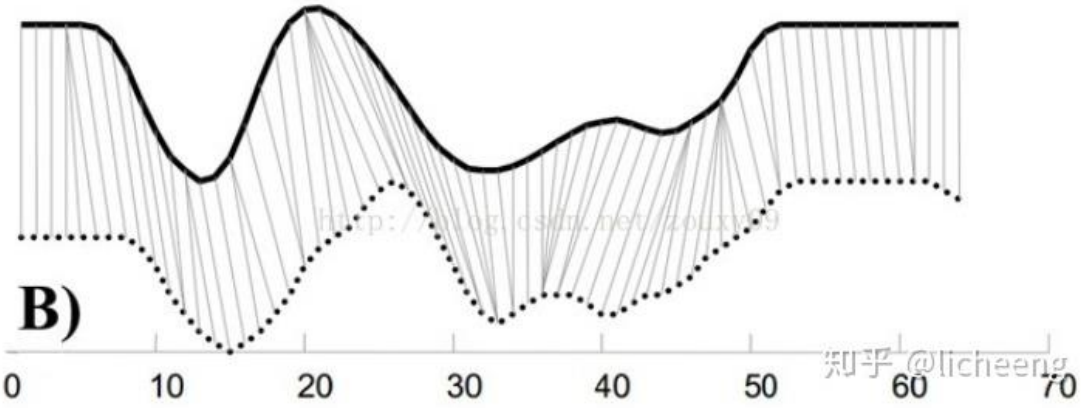

图 17-2 DTW 算法

DTW 算法一般被运用于语音识别、手势识别等。同样的一句话，在不同的说话者口中可能有不同的说话速度、音调高低、节奏急缓，但是所有人对于同一个字的咬字发音特点都是相似的，由此可以对人的说话内容进行判断。

更加抽象的表述是，DTW 算法事实上是在缩放着判断两条曲线的“形状”是否相似。见下图：

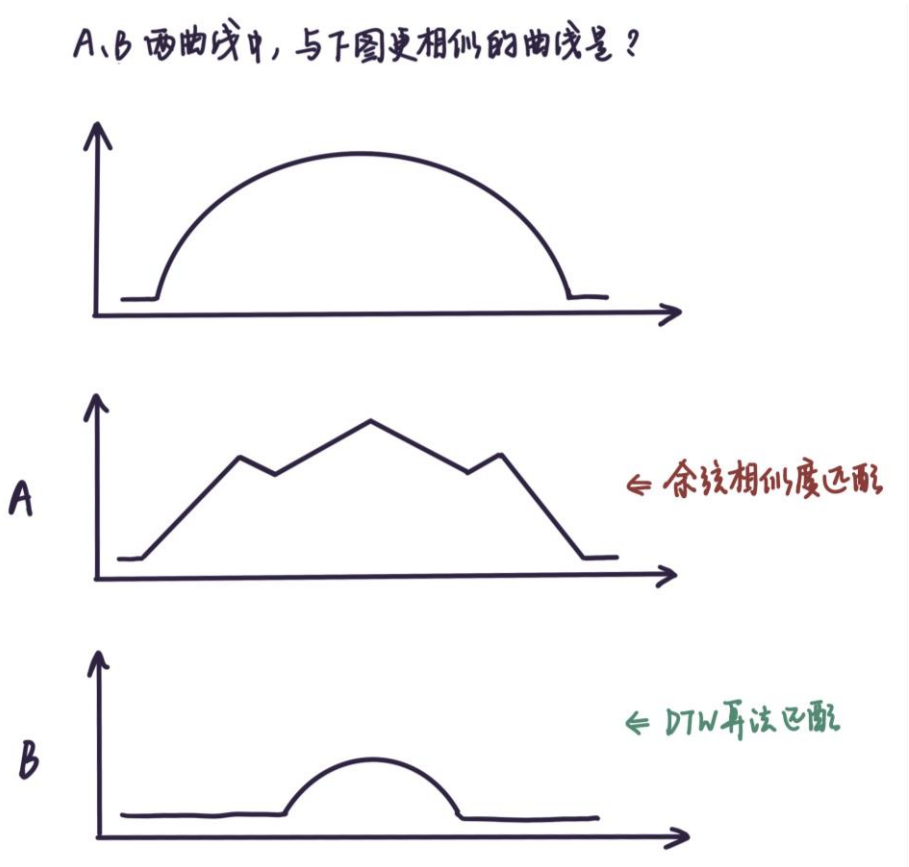

图 18 DTW 算法优势

联系到我们的代码时间习惯相似度匹配，我们认为 DTW 匹配出的相似 cp 是有意义的。

例如，学生 A 因为前期家里有事，实际工作时间段为任务时间后半段；学生 B 因为后期有其他安排，实际工作时间段为任务时间前半段，但是学生 A 和 B 都习惯于“写一天歇一天”的工作模式。由此获得的趋势图很可能因为时间偏移大不相同，但两人实际上的工作模式是一致的。在下一次任务期间，除去了客观时间安排上的影响，两个人很有可能达成一致的步调。

```python
def dtw_distance(ts_a, ts_b, d=lambda x, y: abs(x - y), mww=10000):
    # Create cost matrix via broadcasting with large int
    ts_a, ts_b = np.array(ts_a), np.array(ts_b)
    M, N = len(ts_a), len(ts_b)
    cost = np.ones((M, N))

    # Initialize the first row and column
    cost[0, 0] = d(ts_a[0], ts_b[0])
    for i in range(1, M):
        cost[i, 0] = cost[i - 1, 0] + d(ts_a[i], ts_b[0])

    for j in range(1, N):
        cost[0, j] = cost[0, j - 1] + d(ts_a[0], ts_b[j])

    # Populate rest of cost matrix within window
    for i in range(1, M):
        for j in range(max(1, i - mww), min(N, i + mww)):
            choices = cost[i - 1, j - 1], cost[i, j - 1], cost[i - 1, j]
            cost[i, j] = min(choices) + d(ts_a[i], ts_b[j])

    # Return DTW distance given window
    return cost[-1, -1]
```

图 19 DTW 算法代码

为了降低匹配难度，排除意外情况，我们去除了头尾的数据为零的情况。这样可以直观地获得学生有效工作时间段的代码量变化趋势，减小实际工作时间的偏移对于工作模式相似度的影响。

但是，DTW 的匹配方式虽然精确，也具有不足之处。它仅仅能够匹配出趋势上的相似，而对于互补却并不能匹配。差异度越大并不代表着就是互补而是有可能时间长短完全不同，所以具有模式上的局限性。

在完成 DTW 匹配之后，我们使用 matplotlib 进行了匹配结果的绘制。每张图针对一个学生，把他和他的最佳匹配结果放在一张图中检测匹配的准确性。最终发现除了极个别的异常数据的不匹配现象之外，其他趋势都选择了较为相近的去匹配。这里的异常数据主要是助教的测试数据，在实际运用中，由于这些测试数据只做了一道题，所以并不会被学生匹配到。

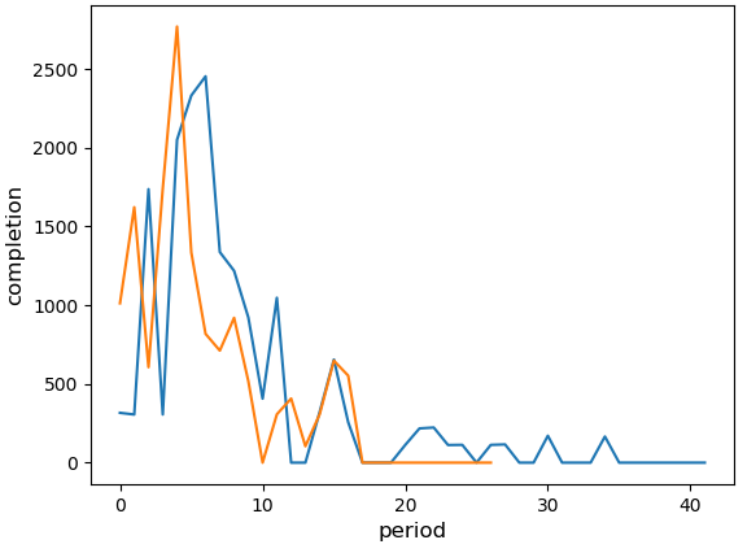

图 20 DTW 算法匹配结果图：60581-60671

C、求导差算法匹配

趋势图匹配的目标是寻找与被匹配对象趋势图走势最为相似（或相反）的趋势图，而这个“走势”如何用数值的形式呈现成为我们需要思考的问题。在前面的研究中我们已对所有趋势图进行非线性回归分析，并获得效果优良的函数拟合结果。拟合函数为趋势折线图的分析提供了可操作性更高的泛化模型，其图像直观的展示了趋势图的走向，由此我们产生了对拟合函数求导的想法。

导数的几何意义为曲线 $y=f(x)$ 在点 $P(x_0, f(x_0))$ 处的切线的斜率 k。若干点处的切线斜率 k 的集合一定程度上可以反映该曲线的“走势”。于是在为两条函数曲线计算相似度时，我们考虑取两者中较短的横轴，在横轴区间内取等距离的 n 个点，计算两个函数导数在这 n 个点的导数值并分别存入两数组。求两数组 n 对导数值之差的平均值作为求导差相似度，导数差越小函数曲线相似程度越高。

```
pairs = []
n = 20  # 设置取点个数
# 大致分类中每两个函数进行比较，取等距离点n个，求两函数在n个点上的导数差之和并取平均得到导数差，导数差越小相似度越高
for item in all:
    pairs_item = {}
    pairs_item["userid"] = item
    pairs_item["companions"] = []
    for item2 in all:
        companion = {}
        deriv1 = []
        deriv2 = []
        del_deriv = []
        companion["companion_id"] = item2
        if item==item2 :
            continue
        if item != item2:
            end = all[item]["end"] if all[item]["end"] <= all[item2]["end"] else all[item2]["end"]
            x = np.linspace(0.5, end, n)
            for i in x:
                deriv1.append(getDeriv(all[item]["拟合类型"], all[item]["coefficient"], i))
                deriv2.append(getDeriv(all[item2]["拟合类型"], all[item2]["coefficient"], i))
                del_deriv.append(
                    abs(getDeriv(all[item]["拟合类型"], all[item]["coefficient"], i) - getDeriv(all[item2]["拟合类型"]
                                 all[item2]["coefficient"], i))),
        companion["similarity"] = np.float(np.mean(del_deriv))
        pairs_item["companions"].append(companion)
```

图 21 求导差相似度算法

求函数导数需要解析先前存有拟合函数信息的 json 文件，取出拟合函数系数并生成相应函数式，再进行求导。

```
def getDeriv(type, coef, i):
    x = symbols('x', real=True)
    # 高斯
    if type == 0:
        a = float(str(coef[0])[0:8])
        b = float(str(coef[1])[0:8])
        c = float(str(coef[2])[0:8])
        d = float(str(coef[3])[0:8])
        y = a * np.power(np.e, -np.square(x - b) / 2 * np.square(c)) + d
        return diff(y, x, 1).subs(x, i)
    # 正弦
    elif type == 1:...
    # 指数
    elif type == 2 or type == 3:...
    # 反比例
    elif type == 4:...
    # 一次
    elif type == 5 or type == 6:...
    # 二次
    elif type == 7 or type == 8:...
```

图 22 解析函数并求导

经过匹配，我们得到每位学生与其他所有学生之间的相似度数据。由此可以得到该学生的最相似和最相反匹配 cp。虽然求导差匹配法由于建立在拟合函

数的基础上而不可避免的会有误差，但从结果图来看，匹配效果依旧达到了预期。

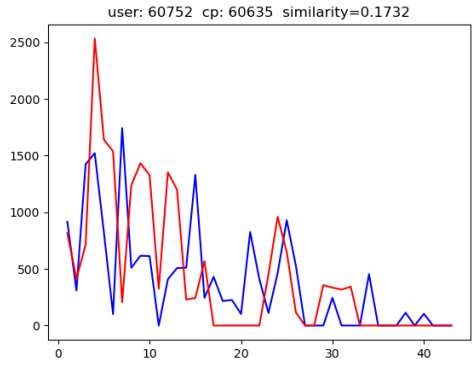

图 23-1 相似 cp 匹配：60752-60635

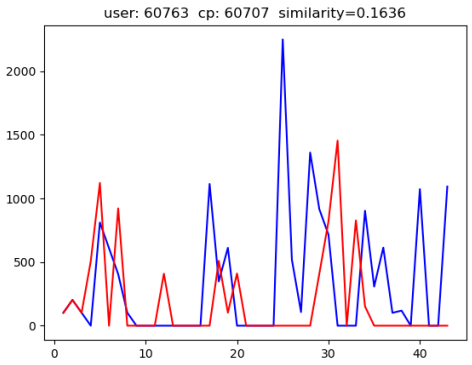

图 23-2 相似 cp 匹配：60763-60707

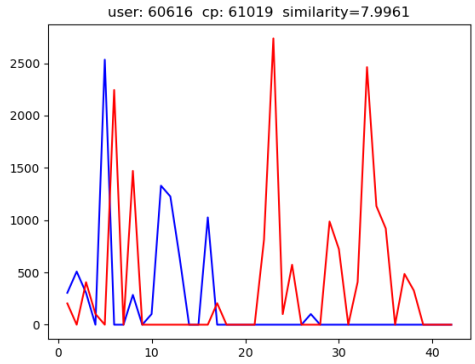

图 23-3 互补 cp 匹配：60616-61019

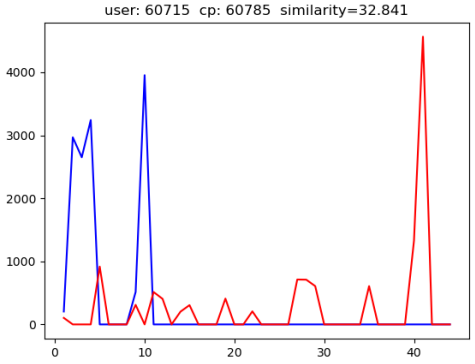

图 23-4 互补 cp 匹配：60715-60785

✧ 雷达图匹配算法

在已经得到雷达图各项能力数值的基础上，我们对于所有学生的雷达图进行了相似或者互补的匹配。

主要的匹配方法是看学生各项能力的差值之和，差值越大证明学生能力分布区域相差越大，而差值小的则证明学生能力相近。最后对于匹配得到的数值存于 json 文件中。

第六阶段：逻辑整理封装

至此，我们的探究内容已经基本完成，但是目前的成果只以数据集、图包的方式展现，还差一个“输入-输出”的接口。为此，我们进行了逻辑的整理和封装，以达到“输入学生 id-获得匹配 cp 列表”的使用效果。

考虑到在实际运用场景下，这个“输入-输出”的过程应当是一个查询过程，而不是低效的计算过程，我们并没有把具体的拟合、匹配等代码逻辑封装进去，而仅仅是实现了“一键查询”的效果。

按照优先趋势图匹配、参考雷达图匹配的设想，我们的 cp 匹配思路为：

1、学生选择趋势图匹配模式：相似/互补

2、后台按照模式进行趋势图匹配，获得候选 cp 列表

3、给列表候选 cp 逐一标记雷达图匹配类型，并返回给学生

4、学生在候选 cp 列表中选择一个“拖延症克星”编程 cp。

同时我们也在思考：除了编程能力在各个维度的偏重，整体编程水平的高低是否也应该被纳入考虑范围内？如果让水平相近的两个人互相竞争，双方都有可能被激发出更大的潜力；如果让猎狗追赶羚羊，即便速度慢上一筹，也能燃起猎狗追赶的斗志；如果要骑车追赶飞机，那只会让人早早放弃，心安理得地被甩在后面。所以，给学生匹配能力相近或是略胜的编程 cp，应该是最合理的做法。

我们将选择编程 cp 能力浮动范围的权力交给学生。学生可以输入两个数值，分别代表以自身能力为基准向下、向上的最大浮动范围。由此，我们的 cp 匹配过程为：

1、学生选择趋势图匹配模式：相似/互补，并输入预期 cp 能力水平（基于自身水平的）上下浮动范围

2、后台按照模式进行趋势图匹配，获得候选 cp 列表

3、将候选 cp 列表中能力不符合预期的候选 cp 删除

4、给列表候选 cp 逐一标记雷达图匹配类型，并返回给学生

5、学生在候选 cp 列表中选择一个“拖延症克星”编程 cp。

```python
if __name__ == '__main__':
    user = User(60639)
    score_range = (0, 10)  # 允许匹配cp的分数差距范围
    # 参数：
    # 每种方法匹配人数，
    # T-相似 F-相反，
    # 是否画图，
    # cp分数范围：[本人分数+score_range[0],本人分数+score_range[1]]
    cp_list = user.getCpList(10, False, False, score_range[0], score_range[1])
    print(len(cp_list))
    print(cp_list.keys())
    for item in cp_list:
        print(item, ":", cp_list[item])
```

图 24 cp 匹配启动器：

```
60762 : {'method': ['求导差匹配'], 'similarities': [3.2223979313187856], 'matchType': ['相似']}
48117 : {'method': ['求导差匹配'], 'similarities': [2.255662089642818], 'matchType': ['相似']}
60896 : {'method': ['余弦相似度匹配'], 'similarities': [0.12439307553087084], 'matchType': ['相似']}
60587 : {'method': ['余弦相似度匹配'], 'similarities': [0.3006428981358011], 'matchType': ['相似']}
```

图 25 匹配成果

3、研究方法总结：

✧ 取样方法：

将 5 组题目提交记录视为**独立**的 5 份样本，取一份进行分析。

✧ 最小二乘法非线性回归分析：

在对做题趋势图进行曲线拟合时，我们采用了最小二乘法非线性回归分析。通过对趋势图进行曲线拟合，可以反映出学生做题的时间安排习惯。拟合的精确度要求并不高，我们将其作为一个**代替人工划分趋势类型的工具**，以便在后续对大规模学生样本进行 cp 匹配时能通过趋势分类降低复杂度。

✧ 余弦相似度匹配:

余弦相似度是对于趋势图**最直观、最直接**的相似匹配方法，匹配结果往往具有**肉眼可见的相似或相异**。

余弦相似度匹配是通过计算两条曲线中对应两向量之间夹角余弦值之和来确定相似度，可以直观地对比出两条曲线的发展趋势是否相似。最后计算出的余弦相似度区间为[0,1]，相似度数值越高，相似度越高。

✧ 动态时间规整（DTW）算法:

DTW（Dynamic Time Warping ）动态时间规整算法，是一种衡量两个长度不同的时间序列的相似度的方法，**可以为我们匹配在“不同的工作时长”中持有“相似的工作模式”，即“相似的时间安排模式”的编程 cp。**

DTW 算法的工作原理可以归结为寻找一条通过此网格中若干格点的路径，路径通过的格点即为两个序列进行计算的对齐的点。路径越长，相似度越低。

✧ 求导差算法:

求导差匹配为我们寻找在代码完成度“变化趋势”，即“积极性”上的编程 cp。

求导差是在将曲线拟合为合适的函数后，通过计算两条曲线函数的导数并在有效区间内取 n 个点，计算 n 对导数值之差的平均值得到求导差相似度。求导差越小相似度越高。

4、研究使用的数据集（图片展示）：

图 26 非法代码过滤后的提交信息

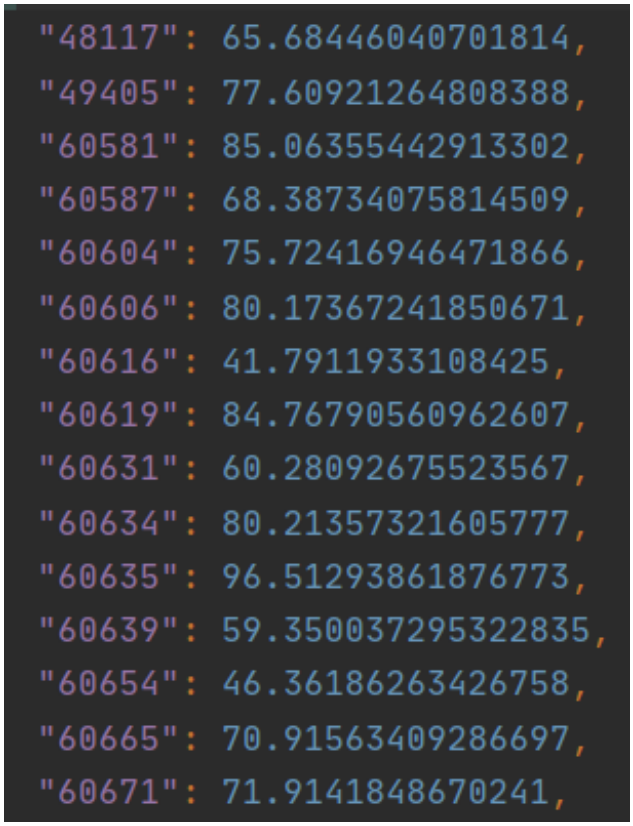

图 27 个人均分信息

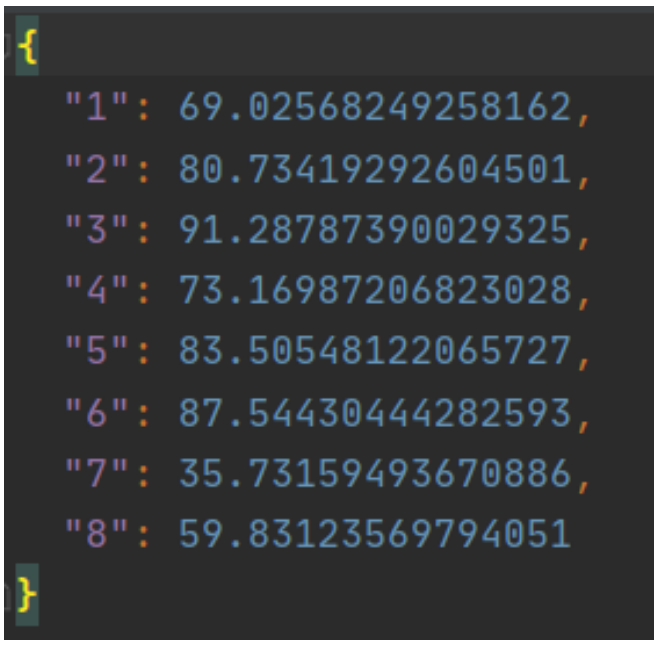

图 28 题型均分信息

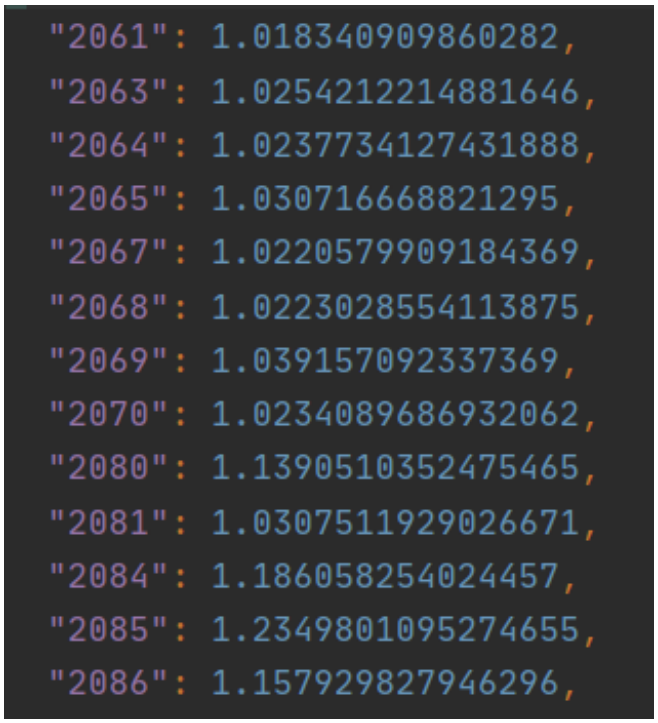

图 29 题目难度系数信息

```
"60619": {
  "拟合类型": 3,
  "拟合函数": "下降指数函数",
  "趋势": "下降",
  "相关系数R": 0.9569684872187324,
  "coefficient": [
    13.311389745136351,
    0.8856743444923088,
    -3.858912480373058
  ],
  "end": 10
},
```


图 30 趋势图曲线拟合信息


```
"上升": [
  "60785"
],
"下降": [
  "48117",
  "60581",
  "60619",
  "60635",
  "60671",
  "60676",
  "60752",
  "60773",
  "60788"
],
"上升-下降": [
  "60606",
  "60616",
  "60762",
  "60899"
],
"下降-上升": [
  "60769",
  "60772",
  "60799",
  "61094"
],
"波动": [
  "49405",
  "60587",
  "60604",
  "60631",
  "60634",
  "60639",
  "60654",
  "60665",
  "60686",
  "60707",
  "60708",
  "60715",
```


图 31 趋势类型分组信息

```
"60581": {
  "companions": [
    {"companion_id": "60785"...},
    {"companion_id": "48117"...},
    {
      "companion_id": "60619",
      "similarity": 0.8524121482975481
    },
    {
      "companion_id": "60635",
      "similarity": 0.6837811356543888
    },
    {
      "companion_id": "60671",
      "similarity": 0.748166518714919
    },
```


图 32 余弦相似度匹配结果


```
"48117": {
  "userid": "48117",
  "companions": [
    {
      "companion_id": "60799",
      "similarity": 0.7631484755458795
    },
    {
      "companion_id": "60752",
      "similarity": 0.7775210199244604
    },
    {
      "companion_id": "60635",
      "similarity": 0.9121185318659271
    },
```


图 33 求导差匹配结果

```
"60785": [
  {
    "companion_id": "61094",
    "similarity": 8459.805368156345
  },
  {
    "companion_id": "60707",
    "similarity": 9438.603099497128
  },
  {
    "companion_id": "60587",
    "similarity": 9516.464556357887
  },
  {
    "companion_id": "60896",
    "similarity": 9652.646265698204
  },
```


图 34 DTW 算法匹配结果


```
"48117": [
  {
    "companion_id": "60671",
    "similarity": 0.908939014202172
  },
  {
    "companion_id": "60639",
    "similarity": 1.1421956296098172
  },
  {
    "companion_id": "60899",
    "similarity": 1.18847624110782
  },
  {
    "companion_id": "60715",
    "similarity": 1.3213202087343965
  },
```


图 35 雷达图匹配结果

图 36 图片文件夹展示

图 37 json 文件夹展示

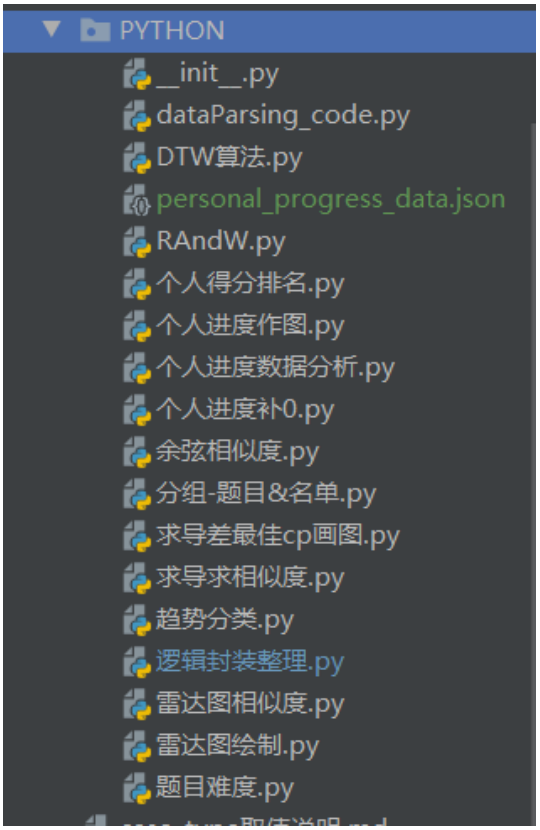

图 38 python 代码文件夹展示

四、案例分析：

在案例分析部分，我们将选择学生 60619、学生 60785，分别对趋势图进行相似、互补匹配，和他们一起完成我们拟合、匹配的全过程。

✧ 学生 60619：

1、用户做题趋势拟合图

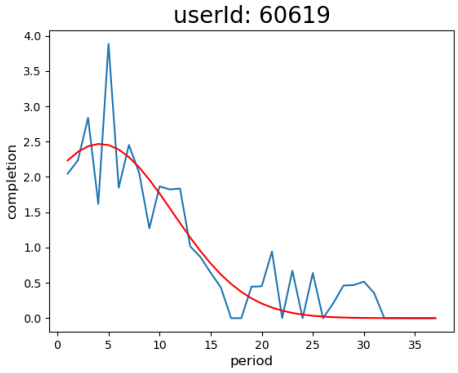

图 39-1 时间单位-1 天

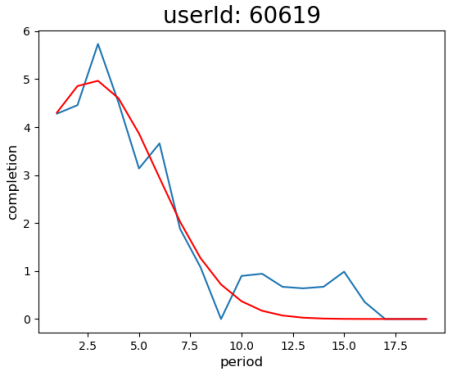

图 39-2 时间单位-2 天

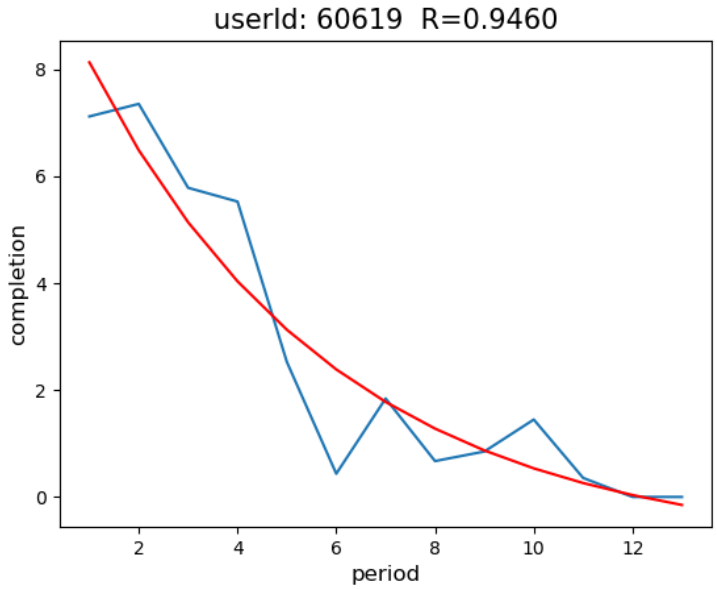

图 39-3 时间单位-3 天，最适拟合图


```
"60619": {
  "拟合类型": 3,
  "拟合函数": "下降指数函数",
  "趋势": "下降",
  "相关系数R": 0.9460684872187324,
  "coefficient": [
    13.311389745136351,
    0.8856743444923088,
    -3.858912480373058
  ],
  "end": 10
},
```


图 39-4 图像拟合信息

由此可见，学生 60619 是一个典型的“好学生”。即便不是“笨鸟”，他也习惯“先飞”，合理的时间安排让他在后期别人都在和 DDL 赛跑的时候泰然自若。

2、用户能力雷达图

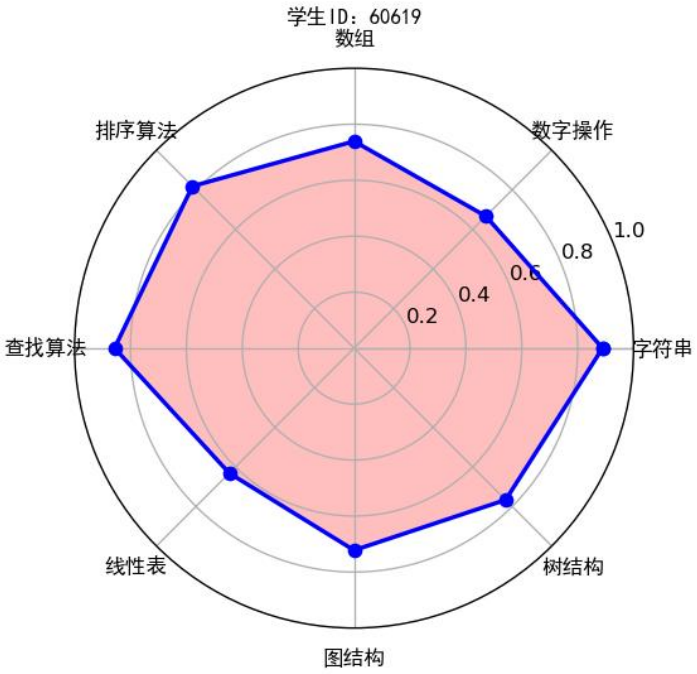

图 40 60619 能力雷达图

60619 的能力雷达图佐证了他的优秀。事实证明，时间安排能力强的人，代码水平自然也不会差。

3、相似 cp 匹配

学生 60619 已经足够优秀了。我们希望为他匹配与他同等优秀的竞争对手，所以选择相似匹配。

我们选择分数浮动范围为[本人分数-10,本人分数+10]，为他筛选能力相近的编程 cp。

```python
if __name__ == '__main__':
    user = User(60619)
    score_range = (-5, 5)  # 允许匹配cp的分数差距范围
    # 参数：
    # 每种方法匹配人数，
    # T-相似 F-相反，
    # 是否画图，
    # cp分数范围：[本人分数+score_range[0],本人分数+score_range[1]]
    cp_list = user.getCpList(2, True, False, score_range[0], score_range[1])
    print(len(cp_list))
    print(cp_list.keys())
    for item in cp_list:
        print(item, ":", cp_list[item])
```

图 41 60619 匹配条件

```
2
dict_keys(['60581', '60737'])
60581 :
匹配方法： ['求导差匹配', '余弦相似度匹配']
雷达图匹配模式： ['相似']
60737 :
匹配方法： ['求导差匹配']
雷达图匹配模式： ['各有所长']
```

图 42 匹配结果

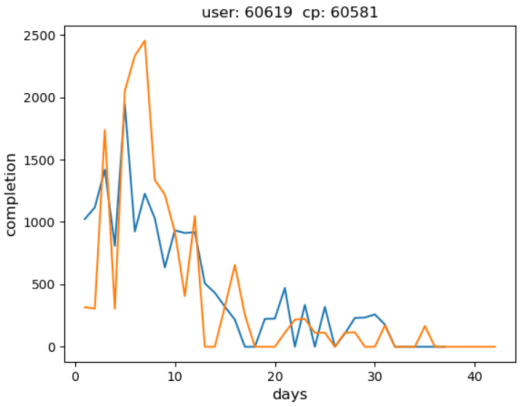

图 43-1 60619-60581 趋势图对比

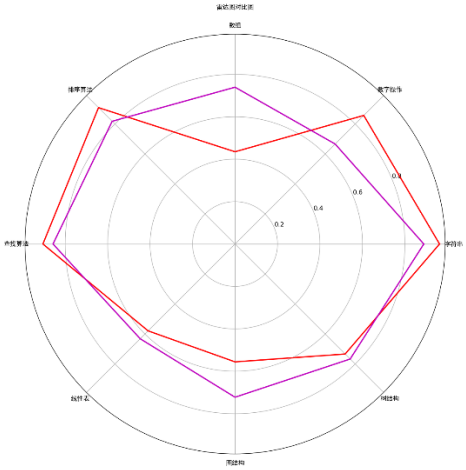

图 43-2 60619-60581 雷达图：相似

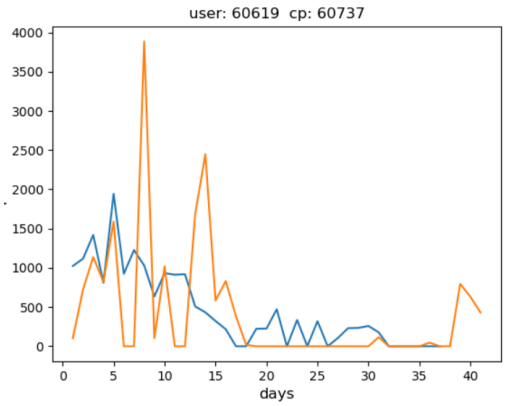

图 43-3 60619-60737 趋势图对比

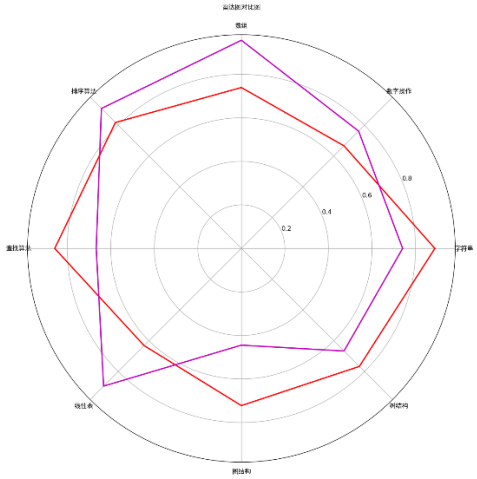

图 43-4 60619-60737 雷达图：各有所长

✧ 学生 60785：

1、用户做题趋势拟合图

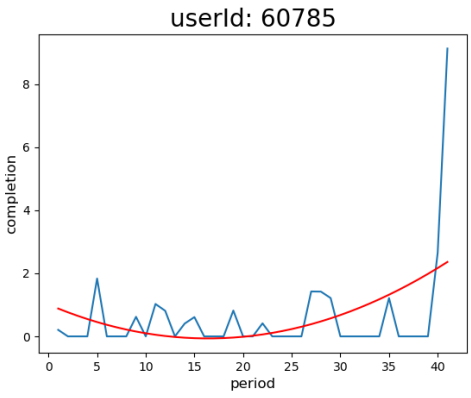

图 44-1 时间单位-1 天

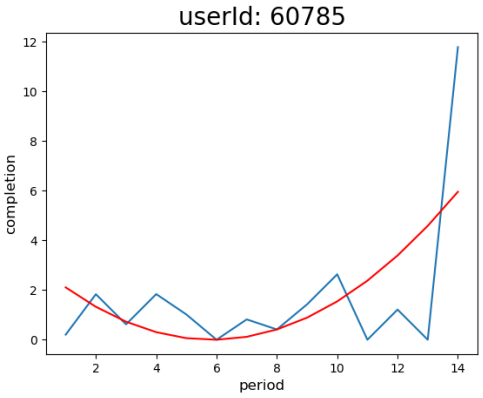

图 44-2 时间单位-3 天

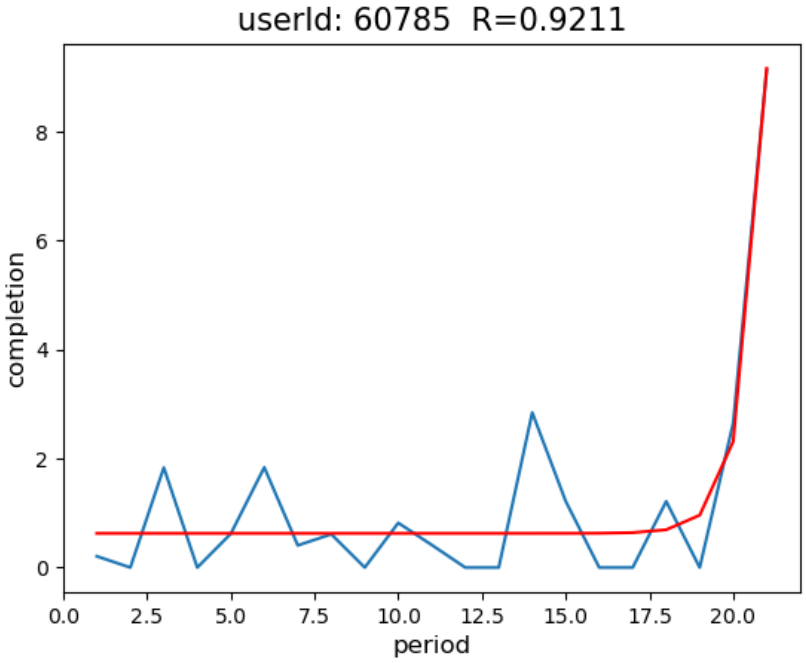

图 44-3 时间单位-2 天，最适拟合图


```
"60785": {
  "拟合类型": 2,
  "拟合函数": "上升指数函数",
  "趋势": "上升",
  "相关系数R": 0.9211842095290149,
  "coefficient": [
    2.0411792990098846e-12,
    5.519447495391022,
    0.7998457001669163
  ],
  "end": 6
},
```


图 44-4 图像拟合信息

这位 60785 同学是我们在探究过程中所见过的，唯一一位有着显著得惊人的指数增长形状做题趋势图的同学。这位同学是我们“拖延症克星”编程 cp 匹配最需要“克”的对象。我们高度怀疑这位同学是否精通面向用例技巧。

2、用户能力雷达图

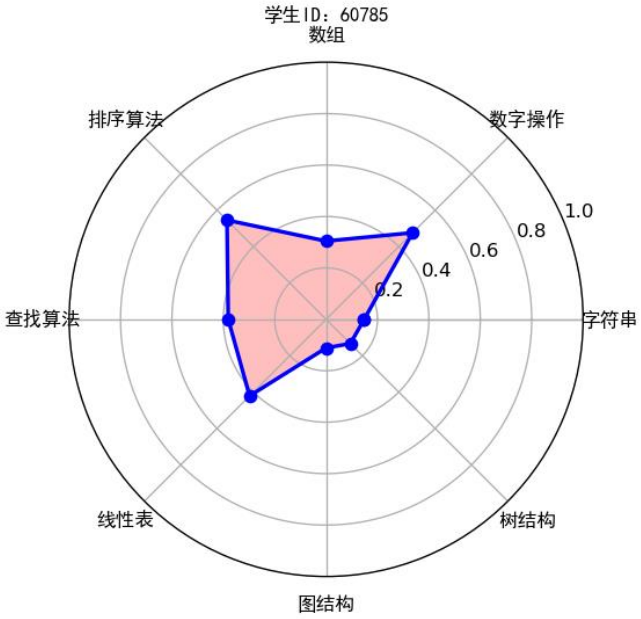

图 45 60785 能力雷达图

根据雷达图，60785 同学在排序、查找、线性表、数字操作等简单类型题目上有稍好的表现。

3、互补 cp 匹配

为了能够督促 60785 同学摆脱“拖延症”的困扰，我们选择趋势图互补的模式为他匹配编程 cp。同时，为了让他可以和更加优秀的同学学习时间安排管理的技巧、咨询题目，参考其本人均分：59 分，我们为他选择的 cp 分数段为[本人分数+10,本人分数+30]。

```python
if __name__ == '__main__':
    user = User(60785)
    score_range = (10,30)  # 允许匹配cp的分数差距范围
    # 参数：
    # 每种方法匹配人数，
    # T-相似 F-相反，
    # 是否画图，
    # cp分数范围：[本人分数+score_range[0],本人分数+score_range[1]]
    cp_list = user.getCpList(3, False, True, score_range[0], score_range[1])
```

图 46 60785 匹配条件

```
3
dict_keys(['60799', '61019', '60829'])
60799 :
匹配方法： ['求导差匹配']
雷达图匹配模式： ['各有所长']
61019 :
匹配方法： ['余弦相似度匹配']
雷达图匹配模式： ['各有所长']
60829 :
匹配方法： ['余弦相似度匹配']
雷达图匹配模式： ['互补']
```

图 47 匹配结果

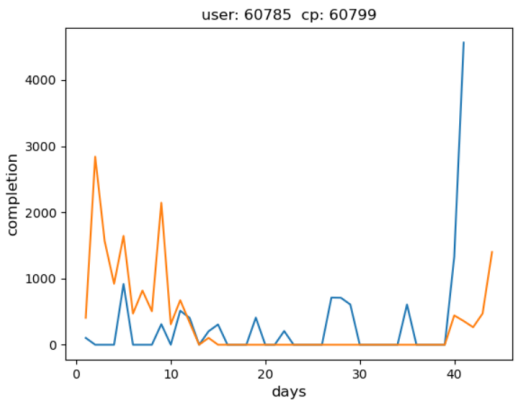

图 48-1 60785-60799 趋势图对比

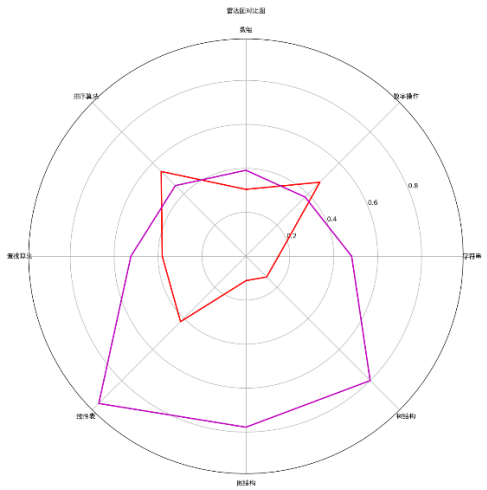

图 48-2 60785-60799 雷达图：各有所长

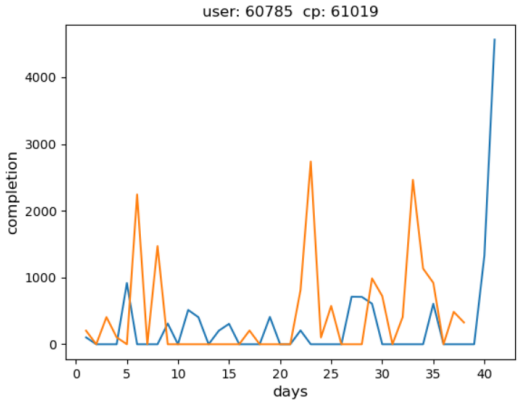

图 49-1 60785-61019 趋势图对比

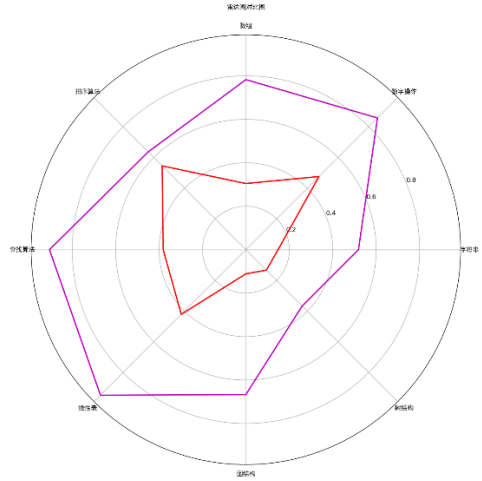

图 49-2 60785-61019 雷达图：各有所长

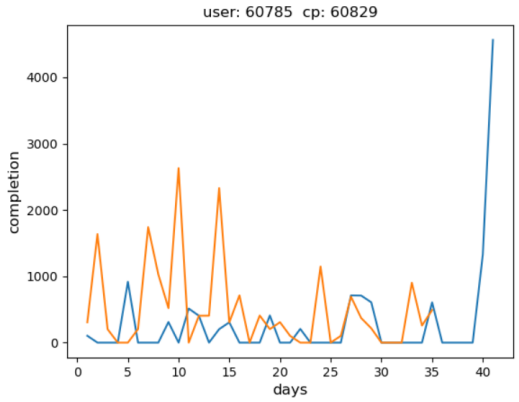

图 50-1 60785-60829 趋势图对比

图 50-2 60785-60829 雷达图：互补

五、附录：

✧ DTW 简介：

来源：https://blog.csdn.net/qq_39516859/article/details/81705010

动态时间规整 DTW 是一个典型的优化问题，它用满足一定条件的时间规整函数 W(n)描述测试模板和参考模板的时间对应关系，求解两模板匹配时累计距离最小所对应的规整函数。

假设我们有两个时间序列 Q 和 C，他们的长度分别是 n 和 m：（实际语音匹配运用中，一个序列为参考模板，一个序列为测试模板，序列中的每个

点的值为语音序列中每一帧的特征值。例如语音序列 Q 共有 n 帧，第 i 帧的特征值（一个数或者一个向量）是 qi。至于取什么特征，在这里不影响 DTW 的讨论。我们需要的是匹配这两个语音序列的相似性，以达到识别我们的测试语音是哪个词）

$$Q = q1, q2,\ldots,qi,\ldots, qn$$

$$C = c1, c2,\ldots, cj,\ldots, cm$$

如果 n=m，直接计算两个序列的距离就好了。但如果 n 不等于 m 我们就需要对齐。最简单的对齐方式就是线性缩放了。把短的序列线性放大到和长序列一样的长度再比较，或者把长的线性缩短到和短序列一样的长度再比较。但是这样的计算没有考虑到语音中各个段在不同情况下的持续时间会产生或长或短的变化，因此识别效果不可能最佳。因此更多的是采用动态规划（dynamic programming）的方法。

为了对齐这两个序列，我们需要构造一个 n x m 的矩阵网格，矩阵元素(i, j)表示 qi 和 cj 两个点的距离 d(qi, cj)（也就是序列 Q 的每一个点和 C 的每一个点之间的相似度，距离越小则相似度越高。这里先不管顺序），一般采用欧式距离，d(qi,cj)=(qi−cj)2d(qi,cj)=(qi−cj)2（也可以理解为失真度）。每一个矩阵元素(i, j)表示点 qi 和 cj 的对齐。DP 算法可以归结为寻找一条通过此网格中若干格点的路径，路径通过的格点即为两个序列进行计算的对齐的点。