

小组信息: 1 人+181840090_黄成东_941147563@qq.com_小于 100+所有工作

研究题目: 统计学习自动定义题目的难度

研究题目:

OJ 平台是许多编程学习者了解熟悉自己对编程掌握情况的重要途径, 对于一个 OJ 平台而言, 题目难度系数的定义尤为重要。

对用户而言, 他能让用户更快确定自己的做题目标, 能够让用户更加有序, 有针对性地进行自身的提高, 完善自己的做题计划, 减少无效时间的浪费。

对于平台而言, 能够更好地将题目进行分类排版规划, 也更好地完成 OJ 平台的更多的功能, 例如评价用户的编程能力, 评价用户的编程水平, 自动挖掘学生的编程缺陷, 自动生成编程学习途径, 自动挖掘最佳编程 CP, 自动推荐代码片段。

如何自动定义题目的难度? 在实际 OJ 平台中, 题目的难度应该是需要随着数据量的完善而变化的, 一开始应该给出一个初始难度系数, 后来题目应该随着做题人数增加而趋向一个稳定的值。

与此同时, OJ 平台的做题机制导致了题目的分差较大, 0 分和 100 分较多, 平均分这种最多时候用来定义难度的数据并不完全适合于 OJ 平台的难度定义。因此, 自动定义题目难度的方法应该能通过结合 OJ 平台特点, 进行动态的难度修正。

代码开源地址: <https://github.com/TomHCD/Big-code-181840090->

Beta-counting: Beta 概率密度函数分布图片

Calculate: 失分率以及区分度计算

download: 用户代码下载

final: 最后样本难度系数计算与 DLK 计算

reliability: 分类题型信度计算

searchAverageScore: 打印出所有用户的指定信息, 信息如下:

“case_id”, “user_id”, “case_type”, “final_score”, “upload_record”,
“upload_id”, “score”

研究方法:

1.数据分析:

将 test_data.json 文件中的数据打印为 student.csv 文件，并导入 Excel 进行数据可视化分析。数据打印格式依次为:

” case_id” , ” case_type” , ” final_score” , ” upload_id” , ” upload_time” , ” score”

```
1 import json
2 import urllib.request, urllib.parse
3 import os
4 import sys
5 import importlib
6 import csv
7
8 importlib.reload(sys)
9
10 f = open('/Users/huangchengdo/Desktop/test_data.json', encoding='utf-8')
11 res = f.read()
12 data = json.loads(res)
13 #print(data)
14
15 k=3544
16 content = open("/Users/huangchengdo/Desktop/Student2.csv", 'w', encoding="utf-8")
17 while k<100000:
18     try:
19         print(data[str(k)]["user_id"])
20         i=0
21         cases = data[str(k)]['cases']
22         scores = data[str(k)]['cases'][i]['upload_records']
23         while i < len(cases):
24             scores = data[str(k)]['cases'][i]['upload_records']
25             for score in scores:
26                 print(cases[i]["case_id"],',',data[str(k)]["user_id"],',',cases[i]["case_type"],',',cases[i]["final_score"],
27                     ',','score',"score"],file=content)
28                 i = i + 1
29             except KeyError:
30                 k=k+1
31             else:
32                 k=k+1
33             finally:
34                 pass
```

case_id	user_id	case_type	final_score	upload_id	upload_time	score		
2908	3544	字符串	40	236494	1.58202E+12	40		
2908	3544	字符串	40	252563	1.58256E+12	0		
2908	3544	字符串	40	252565	1.58256E+12	0		
2908	3544	字符串	40	252576	1.58256E+12	0		
2908	3544	字符串	40	252577	1.58256E+12	0		
2908	3544	字符串	40	252578	1.58256E+12	0		
2172	3544	线性表	100	268885	1.58321E+12	100		
2172	3544	线性表	100	268889	1.58321E+12	100		
2172	3544	线性表	100	268891	1.58321E+12	100		
2172	3544	线性表	100	268892	1.58321E+12	100		
2176	3544	数组	50	234524	1.58195E+12	0		
2176	3544	数组	50	234528	1.58195E+12	50		
2176	3544	数组	50	235147	1.582E+12	50		
2176	3544	数组	50	239384	1.58211E+12	50		
2307	3544	数组	25	283563	1.5842E+12	25		
2804	3544	数组	0	239258	1.5821E+12	0		
2804	3544	数组	0	246142	1.58235E+12	0		
2908	3544	字符串	0	236494	1.58202E+12	40		
2908	3544	字符串	0	252563	1.58256E+12	0		
2908	3544	字符串	0	252565	1.58256E+12	0		
2908	3544	字符串	0	252576	1.58256E+12	0		
2908	3544	字符串	0	252577	1.58256E+12	0		
2908	3544	字符串	0	252578	1.58256E+12	0		
2456	3544	数组	100	256169	1.58272E+12	100		
2456	3544	数组	100	256175	1.58272E+12	100		
2456	3544	数组	100	256176	1.58272E+12	100		
2456	3544	数组	100	256177	1.58272E+12	100		
2456	3544	数组	100	256178	1.58272E+12	100		
2450	3544	查找算法	20	240848	1.58213E+12	20		
2450	3544	查找算法	20	240849	1.58213E+12	20		
2891	8160	数组	100	310658	1.58553E+12	100		
2374	8160	数组	100	310698	1.58553E+12	100		
2890	8160	数组	100	310671	1.58553E+12	100		
2889	8160	数组	100	310656	1.58553E+12	100		
2922	8160	数组	100	310661	1.58553E+12	100		

接着，暂时将 upload_time 此项数据隐藏，通过 Excel 自带排序算法，得到以 case_type 为主 Key，case_id 与 score 为次 Key 排序数据，final_score 因为大多均为 100 分的最终提交成绩，对于题目难度系数参考意义不大，因此选择仅作为一个次要参考因素。

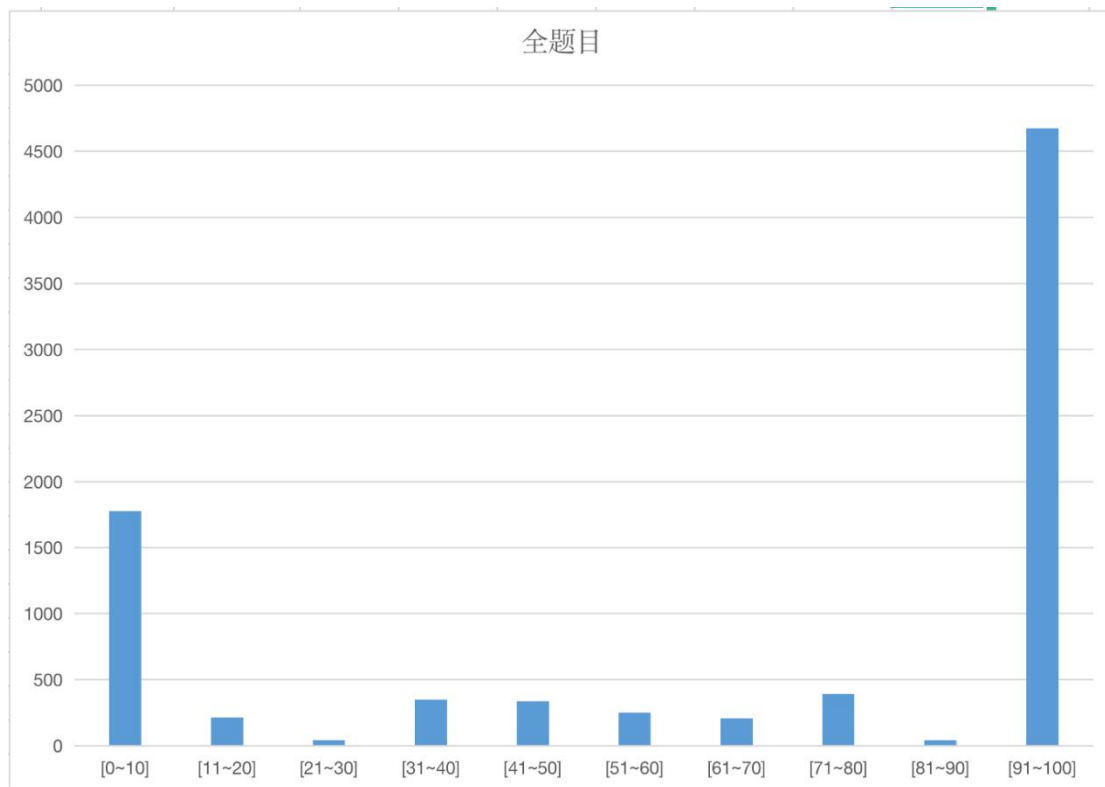


排序后数据:

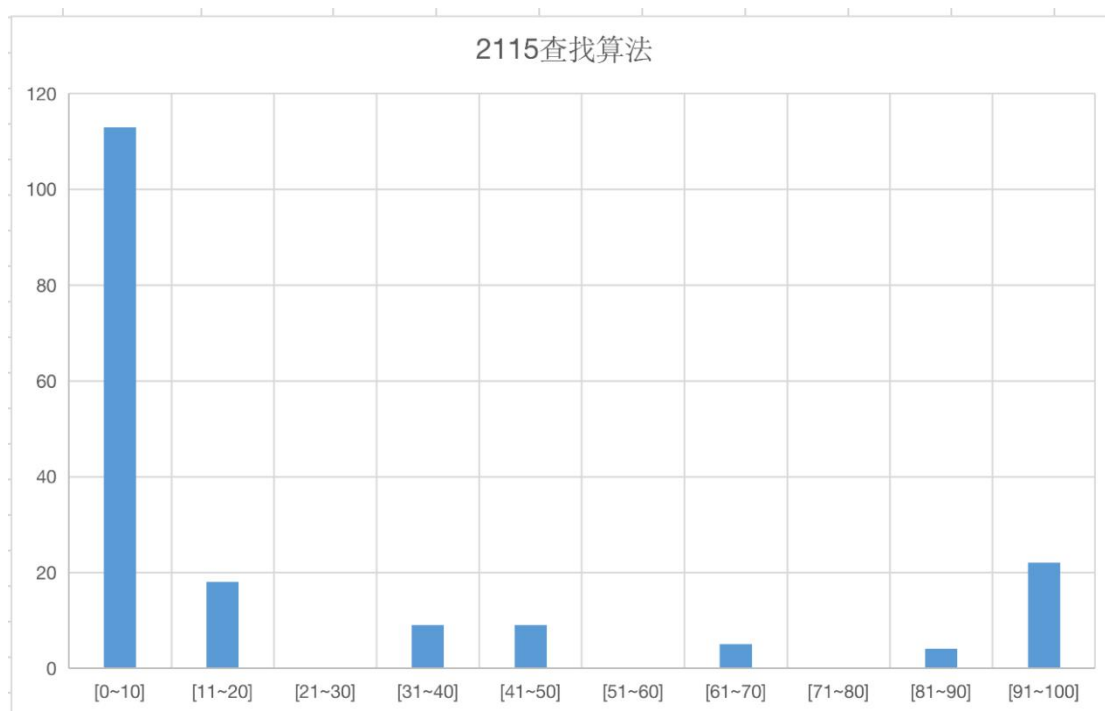
case_id	user_id	case_type	final_score	score			区间分割点	组标识	频率			
2115	60581	查找算法		100	0			10 [0~10]	46			
2115	60581	查找算法		100	0			20 [11~20]	18			
2115	60581	查找算法		100	0			30 [21~30]	0			
2115	60581	查找算法		100	0			40 [31~40]	9			
2115	60581	查找算法		100	0			50 [41~50]	9			
2115	60604	查找算法		0	0			60 [51~60]	0			
2115	60604	查找算法		0	0			70 [61~70]	5			
2115	60619	查找算法		16.67	0			80 [71~80]	0			
2115	60708	查找算法		100	0			90 [81~90]	4			
2115	60708	查找算法		100	0			100 [91~100]	22			
2115	60708	查找算法		100	0							
2115	60708	查找算法		100	0							
2115	60708	查找算法		100	0			区间分割点	组标识	频率		
2115	60708	查找算法		100	0			10 [0~10]	1778			
2115	60708	查找算法		100	0			20 [11~20]	215			
2115	60708	查找算法		100	0			30 [21~30]	44			
2115	60708	查找算法		100	0			40 [31~40]	351			
2115	60708	查找算法		100	0			50 [41~50]	338			
2115	60708	查找算法		100	0			60 [51~60]	252			
2115	60708	查找算法		100	0			70 [61~70]	207			
2115	60708	查找算法		100	0			80 [71~80]	393			
2115	60708	查找算法		100	0			90 [81~90]	45			
2115	60708	查找算法		100	0			100 [91~100]	4675			
2115	60708	查找算法		100	0							
2115	60708	查找算法		100	0							
2115	60708	查找算法		100	0							
2115	60708	查找算法		100	0							
2115	60708	查找算法		100	0							
2115	60708	查找算法		100	0							
2115	60708	查找算法		100	0							
2115	60708	查找算法		100	0							
2115	60708	查找算法		100	0							
2115	60708	查找算法		100	0							
2115	60708	查找算法		100	0							
2115	60762	查找算法		100	0							
2115	60762	查找算法		100	0							
2115	60763	查找算法		100	0							
2115	60763	查找算法		100	0							
2115	60782	查找算法		100	0							
2115	60782	查找算法		100	0							
2115	60782	查找算法		100	0							
2115	60782	查找算法		100	0							
2115	60782	查找算法		100	0							
2115	60782	查找算法		100	0							

通过对所有提交次数进行统计，以 10 为区间进行分割，得到了 10 组区间的频率：

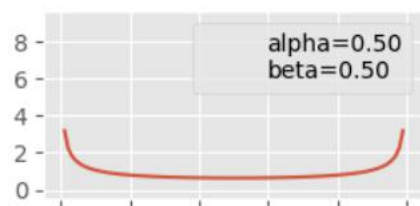
0 ~ 10 为 1778，11 ~ 20 为 215，21 ~ 30 为 44，31 ~ 40 为 351，41 ~ 50 为 338，51 ~ 60 为 252，61 ~ 70 为 207，71 ~ 80 为 393，81 ~ 90 为 45，91 ~ 100 为 4675，进行数据可视化分析以后得到的频率分布直方图为：



同时随机任选一道题目，如 2115 查找算法，进行数据可视化分析得到的频率分布直方图：



实际上从上图可以看出,大部分的 upload_score 基本集中于 0-10 分与 91-100 分,中间段非常少,此时的曲线更加趋向于 beta 分布概率密度函数中 $\alpha < 1$,



beta<1 的情况 (即 U 型曲线), 而非正态分布的情况, 因此使用正态分布曲线进行拟合, 以其通过偏度, 峰值等数据来分析得到题目难度系数是不可靠的, 曲线的两极分化让人想到了一个常用的数据: 区分度, 接下来, 我们将引入失分率, 区分度以及信度这三个指标, 通过这三者来探究与题目难度系数的关系。

2. 难度系数 (G), 失分率 (L), 区分度 (D) 与信度 (B)

难度系数 (G): (理论难度系数+样本难度系数)

在标准化考试系统中, 实体的难度系数是按照目标要求衡量题目难易程度的指标, 它主要依据试题所属的认知水平层次、试题考核的知识面、知识深度、阶梯的推理步数、技能技巧几方面综合评定。

难度系数取值范围为: $0 \leq G \leq 1$, 合理的范围一般为 $0.2 \leq G \leq 0.8$, 太高或者太低的难度系数, 对于检验考生的实际水平很不理想。当 $G=0$ 的时候, 表示试题太简单, 所有考生都从不会失分; 当 $G=1$ 的时候, 表示试题太难, 所有考生都从不会得分。这类试题可以认为是意义不大的题目。

理论难度系数不是专家评定的难度系数, 它一般未知, 但是在一定的环境下 (如: 相同水平的用户, 相同的做题环境, 相同的参考信息), 理论上他应是一个常数, 是衡量用户水平的一把尺子。样本难度系数则是在实际的测试中, 由用户的失分情况计算出来的一个数值, 与试题的失分率有较大的关系。

失分率 (L):

试题的难度系数与题目的失分率有较大关系。失分率高, 表示该题的难度系

数大；否则，难度系数小。但两者又有区别，失分率依赖于考试样本，而难度系数与考试样本关系不大，在一定的环境中，应该是一个比较稳定的数。

设 M 表示题目的满分， n_k 表示第 k 次测试中提交数量， X_k 表示第 k 次测试中全体用户提交的得分之和，则试题在第 k 次测试中的失分率 L_k 和累积失分率 \bar{L}_k 分别为：

$$L_k = 1 - X_k / M * n_k ,$$

$$\bar{L}_k = 1 - [\sum_{i=1, k} X_i / M * \sum_{i=1, k} n_i] ,$$

$$k = 1, 2, \dots$$

区分度 (D) :

区分度是指试题或者试卷对学生实际水平的区分程度或鉴别能力。区分度高的实体或试卷能对不同知识水平和能力的用户加以区分，使能力高的用户得高分，能力低的用户得低分；区分度低的试题则不能对学生的能力进行很好的鉴别，使水平高和水平低的同学得分相差不大或者没有规律可循。

区分度的计算方法有两种：区分度指数和相关系数。

区分度指数：

区分度指数 (index of discrimination, D) 的具体公式如下：

$$D = P_H - P_L$$

式中， D 为区分度指数， P_H 为高分组的项目难度， P_L 为低分组的项目难度。显而易见，高低分两组越是极端，区分度指数就越明显。但个案过少则会减少结果的信度。凯利 (T. L. Kelley) 指出，在正态分布中，兼顾两者的最佳百分数是 27%。对于小样本，如一个常规教学班，可取 25%-33% 之间的任何数字，只要使用方便。

相关系数：

通过计算某一题目得分与测验总得分或效标分数的相关系数来判定。相关越大，区分度越高。

1) 点二列相关

当测验总分为正态连续变量，题目分为二分变量（对、错或通过、未通过）时，可用点二列相关公式计算区分度。其计算公式是：

$$r_{qb} = \frac{\overline{x_p} - \overline{x_q}}{s_t} \times \sqrt{pq}$$

式中, r_{qb} 为二点列相关系数, 即题目区分度; $\overline{x_p}$ 为答对题目被试测验总分平均分, $\overline{x_q}$ 为答错题目被试测验总分平均分, s_t 为全体被试测验总分标准差, p 为答对题目被试占总被试人数比率, $q=1-p$ 。

用点二列相关公式计算出来的相关系数需要进行显著性检验, 才能确定它的意义。检验的方式是对点二列相关公式中 $\overline{x_p}$ 和 $\overline{x_q}$ 的差异进行 t 检验, 若差异显著, 表明 r_{qb} 显著; 若差异不显著, 则 r_{qb} 不显著。

2) 二列相关法

测验总分与题目分两列变量都是正态连续变量, 但其中一列变量由于某种原因被分为两个类别, 可以用二列相关法计算题目区分度。其计算公式是:

$$r_b = \frac{\overline{x_p} - \overline{x_q}}{s_t} \times \frac{pq}{y}$$

式中, r_b 为二列相关系数, 即题目区分度; $\overline{x_p}$ 为答对题目被试测验总分平均分, $\overline{x_q}$ 为答错题目被试测验总分平均分, s_t 为全体被试测验总分标准差, p 为答对题目被试占总被试人数比率, $q=1-p$, y 为正态曲线中答对比例所在位置曲线高度。二列相关系数 r_b 的显著性用下列公式:

$$Z = \frac{r_b}{\frac{1}{y} - \sqrt{\frac{pq}{n}}}$$

式中, r_b 为二列相关系数, p 为答对题目被试占总被试人数比率, $q=1-p$, y 为正态曲线中答对比例所在位置曲线高度。求出 Z 值后, 查正态曲线表, 若 $Z>1.96$, 则相关显著。

3) 四分相关法

四分相关法适用于两列变量都是正态连续变量, 但都要人为地一分为二的统计资料。计算这类相关采用皮尔逊余弦 π 公式, 所得相关为四分相关系数, 公式如下:

$$r_t = \cos \left(\frac{180^\circ}{1 + \sqrt{\frac{AD}{BC}}} \right)$$

式中，A、B、C、D 分别代表四个类别。A 和 D 代表相同符号的次数（++或--），B 和 C 代表相反符号的次数（+-或-+）。

四分相关系数是否显著，可通过下式检验：

$$Z = \frac{r_t}{\frac{1}{y_1 y_2} \sqrt{\frac{p_1 q_1 p_2 q_2}{n}}}$$

式中，p1、q1、p2、q2 为每个类别的累积百分比，y1 和 y2 分别是累计百分比为 p1、p2 时正态曲线的高度，可以通过查正态分布表得到。

运用四分相关计算题目区分度时，样本容量应在 200 以上，计算出的结果才能比较好地说明问题。

4) 积差相关法

对于心理测验中的多值评分的题目和学科测验中的主观性试题，可以用积差相关法计算题目分和测验总分的相关系数，作为题目区分度值。

因为我们可以把在第 k 次考试中分数从高到低排序，取前 27%提交作为高分组，取后 27%作为低分组，分别计算高分组提交的平均成绩 X_{yh} 和低分组学生的平均成绩 X_{yl} ，试题 y 在第 k 次考试中的区分度 D_{yk} 为：

$$D_{yk} = (X_{yh} - X_{yl}) / M,$$

$$k = 1, 2, \dots$$

一般认为， $D > 0.4$ 的试题区分度为“优秀”； $0.3 \leq D \leq 0.4$ 的试题区分度为“良好”； $0.2 \leq D \leq 0.3$ 的试题区分度为“可以”； $D < 0.2$ 的试题区分度为“较差”

信度 (B)：

信度即度即测试结果的可信程度，是反映测试结果一致性，可靠性及稳定性程度的指标。

测试目的是希望得到接近真值的实测值，但二者必然存在误差，误差越小，

信度越高，通常会用相关系数来估计测验的信度。例如对同一组对象施测两次所得的两组成绩的相关系数作为度量信度的指标，此时相关系数也就是测验的信度系数。

常用的信度系数有四种：再测信度系数，复本信度系数，内部一致性系数，评分者信度系数。

我们需要对前三种信度依次进行分析，以其确定哪一种信度更加适合我们。

(第四种因为缺少评分者的条件，因此不予讨论)

1、再测信度系数

再测信度系数也成为稳定性系数，是指在先后两个不同时间内用相同测验对同一组被测对象进行两次施测所得分数的相关系数，简单说就是用相同试卷对相同的人在不同时间测试两次，再测信度系数的计算公式如下：

$$r_{A_1 A_2} = \frac{n \sum_1^n X_i Y_i - \sum_1^n X_i \sum_1^n Y_i}{\sqrt{n \sum_1^n X_i^2 - (\sum_1^n X_i)^2} \sqrt{n \sum_1^n Y_i^2 - (\sum_1^n Y_i)^2}}$$

其中， X_i 是第一次测试成绩， Y_i 是第二次测试成绩， n 为被测试对象的人数。这种方法称为重测法，容易受两次测试时间间隔的影响，间隔太长或太短，都无法说明在第一次测试之后，由于经验、记忆、情绪等诸多因素的变化，会对第二次测试结果造成何种影响，因此不太建议使用。

2、复本信度系数

当同一个测验不合适重复做两次时，可以考虑使用该测验的复本进行测试，复本要求在题型、数量、格式、难度等方面都要与原测试保持一致。当第一次测试完成后，再最短时间间隔内进行第二次测验，再计算两次测验结果的相关系数，即复本信度系数，计算公式跟重测法的公式一致，当复本信度系数较高 (≥ 0.7)，且平均数和标准差比较接近时，认为两次测验等值，这种方法称为复本法。

使用复本法计算信度系数，可以避免重测法受时间间隔影响的确定，但要求复本必须与原测试等值，且测试对象连续进行两次测试时，可能会因为测验时间过长而产生厌倦心理，从而影响测试结果。

3、内部一致性系数

α 系数 (Cronbach)

当测试题目包含主观题时，一般使用克伦巴赫 α 系数作为内部一致性信度系数，公式如下：

$$\alpha = \frac{K}{K-1} \left[1 - \frac{\sum_{i=1}^K S_i^2}{S_t^2} \right]$$

其中 K 为测试的题目数量， S_i^2 为第 i 题各测试者得分的方差， S_t^2 为个测试者所得总分的方差。

我们的数据库并不支持能够进行两次重复的测量，难以达到以上两种方法的要求，因此需要舍弃前两种方法。对于第三种方法而言，实际上需要多道题目的参与，在这个情景下，我们假设同一个知识点试题的信度相同，以此来进行计算。

第 k 次考试中相同知识点信度 B_k 的计算公式为：

$$B_k = \left(\frac{t}{t-1} \right) * \left(1 - \left(\sum_{i=1}^t S_i^2 / S^2 \right) \right),$$

$$k = 1, 2, \dots$$

其中， B_k 表示该知识点试题的信度， t 表示试题总数， S_i 表示第 i 题标准差， S^2 表示试题方差。

一般认为， $B < 0.5$ 的试题信度较差； $0.5 \leq B \leq 0.8$ 的试题信度良好； $B > 0.8$ 的试题信度非常好

3. 样本难度系数修正算法

影响试题样本难度系数的因素主要有：失分率、区分度和信度。

在实际中，如何根据样本数据，计算试题的样本难度系数，进而估计理论难度系数呢？

3.1 统计学习

统计学习的目的是根据给定的训练样本对某系统输入输出之间的依赖关系的估计，使他能够对位置输出作出尽可能准确的估计和预测。可以一般地表示为：样本系数难度 g 与考试变量 $x = (L, D, B)$ 存在一定的未知依赖关系，即遵循某一未知的联合概率 $F(x, g)$ ，其中 L 、 D 、 B 分别表示考试样本的失分率、区分度和信度。统计学习问题就是根据 n 个独立同分布观测样本 (x_1, w_1) ， $(x_2,$

$w_2)$, , (x_n, w_n) , 其中 $w_i = (g_0, g_1, \dots, g_{i-1})'$, g_0 为试题样本系数初始值, 一般由出题者/专家给定, 在没有的情况下, 可以通过在网上寻找旧有题目难度系数或者取初始数据失分率, g_i 为试题第 i 次考试样本难度系数值, 在一组函数 $\{g=f(x, w)\}$ 中求一个最优的函数 $f_0(x, w)$ 对依赖关系进行估计, 使期望风险:

$$R(w) = \int L(G, F(x, w)) dF(x, w)$$

最小。其中 $\{f(x, w)\}$ 称作预测函数集, $g=f(x, w)$ 为试题的样本难度系数, G 为试题的理论难度系数, 在这里为常熟, $\{f(x, w)\}$ 可以表示任何函数集; $L(G, F(x, w))$ 为由于用 $f(x, w)$ 对 G 进行预测而造成的损失, 可以定义如下:

$$L(G, F(x, w)) = (G - f(x, w))^2$$

3.2 经验风险最小化

由于已知的信息有限, $R(w)$ 的期望风险无法计算, 传统的统计学习方法中采用了所谓经验风险最小化 (Empirical Risk Minimization, ERM) 准则, 即用样本定义经验风险

$$R_{\text{emp}}(g) = (1/n) \sum_{i=1, n} L(G, f(x_i, w_i)) = (1/n) \sum_{i=1, n} (G - g_i)^2$$

其中 G 为试题的理论难度系数, x_i 为考试观测样本, $g_i=f(x_i, w_i)$ 为第 i 次考试中试题的样本难度系数, $i=1, 2, \dots, n$ 。

ERM 准则是一般性的, 利用概率密度估计中的最大似然法可以证明: 用样本难度系数 g_i 的统计量去估计理论难度系数, $R_{\text{emp}}(g)$ 能使损失 $L(G, f(x, w))$ 最小。

由最大似然法知, 两边对 g 求偏导, 得

$$d R_{\text{emp}}(g) / d g = (2/n) \sum_{i=1, n} (G - g_i) = 0$$

进而得 $G = (1/n) \sum_{i=1, n} g_i = \text{average}(g)$, 即理论难度系数等于样本难度系数的平均值, 这表明, 试题的理论难度系数虽然未知, 但样本难度系数是其无偏估计。

3.3 样本难度系数序列 $\{G_k\}$ 的收敛性

序列 $\{G_k\}$ 的收敛性是准确计算理论难度系数的关键。为了从理论上探讨预测函数 $g=f(x, w)$ 的结构和性质, 常常考虑用历史资料 g_1, g_2, \dots, g_n 的线性组合 $a_1g_1+a_2g_2+\dots+a_ng_n$ 对未来的 g_{n+k} 进行预测和估计。

设 G_1, G_2, \dots, G_n 为试题的样本难度系数随机变量序列, 令

$$L^2(X) = \{\sum_{j=0, k} a_j G_j \mid a_j \in \mathbb{R}, k \in \mathbb{N}_+\}$$

容易证明, $L^2(X)$ 是线性空间。在 $L^2(X)$ 上定义内积: $\langle X, Y \rangle = E(XY)$, 如此可以证明: $L^2(X)$ 是内积空间, 又是距离空间。从而进一步有: 对任意 $\xi_n \in L^2(X)$, 总有 $\xi_0 \in L^2(X)$, 使得 $\lim_{n \rightarrow \infty} \|\xi_n - \xi_0\| = 0$ 。

因此, $L^2(X)$ 是 Hilbert 空间。由 Hilbert 空间的性质知, 任意 $\xi_n \in L^2(X)$, 基本序列 $\{\xi_n\}$ 必收敛于 $L^2(X)$ 中一个唯一确定的常数 ξ_0 , 这个常数就是某试题的理论难度系数。

这里, 预测函数 $g_k = f(x_k, w_k)$ 在统计学习上也称学习器。

3.4 学习器 $g_k = f(x_k, w_k)$ 的构造

由上面的讨论知, 只要序列 $\{g_k = f(x_k, w_k)\}$ 在 $L^2(X)$ 上收敛, 并且满足 ERM 准则, 则一定可以通过有限的小样本数据, 近似求得试题的理论难度系数 G 。令

$$g_k = f(x_k, w_k) = [\sum_{i=0, k-1} g_i] + \psi(L_k) / k+1, \\ k = 1, 2, \dots$$

其中 $\phi(L_k) = L_k * (D_{yk}/0.4) * (B_k/0.8)$, 若 $\phi(L_k) \geq 1$ 或 $\phi(L_k) = 0$, 则 $\psi(L_k) = \bar{L}_k$, 否则 $\psi(L_k) = \phi(L_k)$, g_k 、 D_{yk} 、 L_k 、 \bar{L}_k 分别为试题 y 在第 k 次考试中的样本难度系数、区分度、失分率和累积失分率, B_k 为第 k 次考试试卷信度。

由 g_k 知, 学习器具有根据它的历史资料进行学习和修正的功能, 使其收敛效果较好。

因为没有专家评定, 因此, 我们决定令题目的:

$$g_0 = (1 - \text{初试平均分}/100)/2$$

此数值较为合理

4. 其他相关因素分析:

1. 相关假设

1. 每个提交的同学的各方面条件相同
2. 题目类别已经确定
3. 去除时间跨度的影响

- 4.题干长短与题目难度没有关系
 - 5.无需使用代码分析
 - 6.题目答案不作为难度分析的标准
 - 7.所有类别的题目服从于一种分布
 - 8.认为同一类题目的信度是相同且不变的
- ## 2.提交时间，代码详情，题目详情为何未被使用

1.提交时间未使用:

实际上，不同同学代码习惯不同，有的同学遇到难的题目以后，会选择放在一边下次再写，这中间造成了很长的时间间隔，与实际做题时间不符合

2.代码详情未使用:

代码的长度很多时候能代表一个项目的复杂程度，例如浏览器、操作系统等原因导致项目的复杂而导致难度增加。然而算法题并未达到该种程度，实际上大部分算法题难度在于思路而非代码量上，因此选择不计入代码详情。

3.题目详情未使用:

题目的知识点、长度相关与题目难度并无直接关系，不同类型题目之所以出现与难度系数部分相关是因为出题者的喜好导致的，并不能作为客观标准，因此题目的详情与题目理论难度系数并无直接关联

案例分析:

我们选取 `case_type` 为查找算法, `case_id` 为 2115 的题目进行难度分析

1.筛选出相关数据:

case_id	user_id	case_type	final_score	upload_id	upload_time	score			
2115	49405	查找算法	100	292013	1.58459E+12	100			
2115	60581	查找算法	100	249736	1.58245E+12	0			
2115	60581	查找算法	100	249753	1.58245E+12	0			
2115	60581	查找算法	100	250014	1.58246E+12	0			
2115	60581	查找算法	100	250020	1.58246E+12	0			
2115	60581	查找算法	100	250029	1.58246E+12	0			
2115	60581	查找算法	100	252223	1.58255E+12	100			
2115	60604	查找算法	0	244804	1.58229E+12	0			
2115	60604	查找算法	0	245490	1.58234E+12	0			
2115	60606	查找算法	100	306883	1.58537E+12	100			
2115	60619	查找算法	16.67	253033	1.5826E+12	16.67			
2115	60619	查找算法	16.67	278750	1.58389E+12	16.67			
2115	60619	查找算法	16.67	278935	1.58391E+12	16.67			
2115	60619	查找算法	16.67	278967	1.58392E+12	0			
2115	60619	查找算法	16.67	278970	1.58392E+12	16.67			
2115	60631	查找算法	100	246261	1.58236E+12	16.67			
2115	60631	查找算法	100	286791	1.58436E+12	100			
2115	60634	查找算法	100	290768	1.58453E+12	100			
2115	60635	查找算法	100	262559	1.58297E+12	100			
2115	60671	查找算法	100	282337	1.58416E+12	50			
2115	60671	查找算法	100	282342	1.58416E+12	100			
2115	60676	查找算法	100	295590	1.58477E+12	100			
2115	60686	查找算法	100	316442	1.58564E+12	100			
2115	60708	查找算法	100	302225	1.5851E+12	0			
2115	60708	查找算法	100	302237	1.5851E+12	0			
2115	60708	查找算法	100	302239	1.5851E+12	0			
2115	60708	查找算法	100	302240	1.5851E+12	0			
2115	60708	查找算法	100	302241	1.5851E+12	16.67			
2115	60708	查找算法	100	302242	1.5851E+12	0			
2115	60708	查找算法	100	302244	1.5851E+12	0			
2115	60708	查找算法	100	302246	1.5851E+12	0			
2115	60708	查找算法	100	302247	1.5851E+12	0			
2115	60708	查找算法	100	302248	1.5851E+12	0			
2115	60708	查找算法	100	302249	1.5851E+12	0			
2115	60708	查找算法	100	302250	1.5851E+12	0			

2115	60708	查找算法	100	302250	1.5851E+12	0			
2115	60708	查找算法	100	302254	1.58511E+12	0			
2115	60708	查找算法	100	302255	1.58511E+12	0			
2115	60708	查找算法	100	302256	1.58511E+12	0			
2115	60708	查找算法	100	302257	1.58511E+12	0			
2115	60708	查找算法	100	302258	1.58511E+12	0			
2115	60708	查找算法	100	302259	1.58511E+12	0			
2115	60708	查找算法	100	302260	1.58511E+12	0			
2115	60708	查找算法	100	302261	1.58511E+12	0			
2115	60708	查找算法	100	302262	1.58511E+12	0			
2115	60708	查找算法	100	302263	1.58511E+12	0			
2115	60708	查找算法	100	302264	1.58511E+12	16.67			
2115	60708	查找算法	100	302265	1.58511E+12	33.33			
2115	60708	查找算法	100	302266	1.58511E+12	33.33			
2115	60708	查找算法	100	302268	1.58511E+12	50			
2115	60708	查找算法	100	302269	1.58511E+12	0			
2115	60708	查找算法	100	302270	1.58511E+12	33.33			
2115	60708	查找算法	100	302272	1.58511E+12	50			
2115	60708	查找算法	100	302273	1.58511E+12	16.67			
2115	60708	查找算法	100	302274	1.58511E+12	16.67			
2115	60708	查找算法	100	302275	1.58511E+12	16.67			
2115	60708	查找算法	100	302276	1.58511E+12	50			
2115	60708	查找算法	100	302277	1.58511E+12	0			
2115	60708	查找算法	100	302278	1.58511E+12	16.67			
2115	60708	查找算法	100	302280	1.58511E+12	33.33			
2115	60708	查找算法	100	302281	1.58511E+12	50			
2115	60708	查找算法	100	302282	1.58511E+12	50			
2115	60708	查找算法	100	302284	1.58511E+12	66.67			
2115	60708	查找算法	100	302285	1.58511E+12	83.33			
2115	60708	查找算法	100	302286	1.58511E+12	0			
2115	60708	查找算法	100	302288	1.58511E+12	100			
2115	60752	查找算法	100	278202	1.58382E+12	100			
2115	60762	查找算法	100	262698	1.58297E+12	0			
2115	60762	查找算法	100	262699	1.58297E+12	0			
2115	60762	查找算法	100	305647	1.5853E+12	100			
2115	60762	查找算法	100	305648	1.5853E+12	0			

2115	60762	查找算法	100	305647	1.5853E+12	100		
2115	60763	查找算法	100	298283	1.58489E+12	0		
2115	60763	查找算法	100	298284	1.58489E+12	0		
2115	60763	查找算法	100	298285	1.58489E+12	100		
2115	60773	查找算法	100	307787	1.58541E+12	100		
2115	60775	查找算法	100	320881	1.58567E+12	100		
2115	60782	查找算法	100	304881	1.58524E+12	16.67		
2115	60782	查找算法	100	311648	1.58555E+12	0		
2115	60782	查找算法	100	311654	1.58555E+12	0		
2115	60782	查找算法	100	311656	1.58555E+12	0		
2115	60782	查找算法	100	311660	1.58555E+12	0		
2115	60782	查找算法	100	311663	1.58555E+12	0		
2115	60782	查找算法	100	311665	1.58555E+12	0		
2115	60782	查找算法	100	311668	1.58555E+12	16.67		
2115	60782	查找算法	100	311674	1.58555E+12	33.33		
2115	60782	查找算法	100	311676	1.58555E+12	50		
2115	60782	查找算法	100	311677	1.58555E+12	66.67		
2115	60782	查找算法	100	311682	1.58555E+12	83.33		
2115	60782	查找算法	100	311687	1.58555E+12	100		
2115	60799	查找算法	16.67	319756	1.58566E+12	16.67		
2115	60799	查找算法	16.67	319811	1.58566E+12	16.67		
2115	60812	查找算法	100	271866	1.58332E+12	100		
2115	60829	查找算法	100	292597	1.58461E+12	0		
2115	60829	查找算法	100	306874	1.58537E+12	100		
2115	60832	查找算法	100	309545	1.58548E+12	100		
2115	60833	查找算法	16.67	319933	1.58566E+12	16.67		
2115	60870	查找算法	100	314014	1.58558E+12	0		
2115	60870	查找算法	100	314019	1.58558E+12	0		
2115	60870	查找算法	100	314027	1.58558E+12	0		
2115	60870	查找算法	100	314030	1.58558E+12	16.67		
2115	60870	查找算法	100	314033	1.58558E+12	33.33		
2115	60870	查找算法	100	314036	1.58558E+12	33.33		
2115	60870	查找算法	100	314044	1.58558E+12	50		
2115	60870	查找算法	100	314049	1.58558E+12	50		
2115	60870	查找算法	100	314055	1.58558E+12	66.67		
2115	60870	查找算法	100	314057	1.58558E+12	66.67		

2115	60782	查找算法	100	311660	1.58555E+12	0		
2115	60782	查找算法	100	311663	1.58555E+12	0		
2115	60782	查找算法	100	311665	1.58555E+12	0		
2115	60782	查找算法	100	311668	1.58555E+12	16.67		
2115	60782	查找算法	100	311674	1.58555E+12	33.33		
2115	60782	查找算法	100	311676	1.58555E+12	50		
2115	60782	查找算法	100	311677	1.58555E+12	66.67		
2115	60782	查找算法	100	311682	1.58555E+12	83.33		
2115	60782	查找算法	100	311687	1.58555E+12	100		
2115	60799	查找算法	16.67	319756	1.58566E+12	16.67		
2115	60799	查找算法	16.67	319811	1.58566E+12	16.67		
2115	60812	查找算法	100	271866	1.58332E+12	100		
2115	60829	查找算法	100	292597	1.58461E+12	0		
2115	60829	查找算法	100	306874	1.58537E+12	100		
2115	60832	查找算法	100	309545	1.58548E+12	100		
2115	60833	查找算法	16.67	319933	1.58566E+12	16.67		
2115	60870	查找算法	100	314014	1.58558E+12	0		
2115	60870	查找算法	100	314019	1.58558E+12	0		
2115	60870	查找算法	100	314027	1.58558E+12	0		
2115	60870	查找算法	100	314030	1.58558E+12	16.67		
2115	60870	查找算法	100	314033	1.58558E+12	33.33		
2115	60870	查找算法	100	314036	1.58558E+12	33.33		
2115	60870	查找算法	100	314044	1.58558E+12	50		
2115	60870	查找算法	100	314049	1.58558E+12	50		
2115	60870	查找算法	100	314055	1.58558E+12	66.67		
2115	60870	查找算法	100	314057	1.58558E+12	66.67		
2115	60870	查找算法	100	314061	1.58558E+12	66.67		
2115	60870	查找算法	100	314066	1.58558E+12	83.33		
2115	60870	查找算法	100	314068	1.58558E+12	83.33		
2115	60870	查找算法	100	314075	1.58558E+12	100		
2115	60885	查找算法	100	311890	1.58556E+12	100		
2115	60896	查找算法	33.33	300777	1.58502E+12	33.33		
2115	61053	查找算法	16.67	306870	1.58537E+12	16.67		
2115	61094	查找算法	33.33	319449	1.58566E+12	33.33		
2115	61406	查找算法	100	313147	1.58557E+12	100		

2.目前的查找算法仍为随机排列，按照 25%-50%-75%-100%分成四个工作序列

(演示中将选取 100%作为展示)

1	case_id	user_id	case_type	final_score	score		
2	2115	60581	查找算法	100	0		
3	2115	60581	查找算法	100	0		
4	2115	60581	查找算法	100	0		
5	2115	60581	查找算法	100	0		
6	2115	60581	查找算法	100	0		
7	2115	60604	查找算法	0	0		
8	2115	60604	查找算法	0	0		
9	2115	60619	查找算法	16.67	0		
10	2115	60708	查找算法	100	0		
11	2115	60708	查找算法	100	0		
12	2115	60708	查找算法	100	0		
13	2115	60708	查找算法	100	0		
14	2115	60708	查找算法	100	0		
15	2115	60708	查找算法	100	0		
16	2115	60708	查找算法	100	0		
17	2115	60708	查找算法	100	0		
18	2115	60708	查找算法	100	0		
19	2115	60708	查找算法	100	0		
20	2115	60708	查找算法	100	0		
21	2115	60708	查找算法	100	0		
22	2115	60708	查找算法	100	0		
23	2115	60708	查找算法	100	0		
24	2115	60708	查找算法	100	0		
25	2115	60708	查找算法	100	0		
26	2115	60708	查找算法	100	0		
27	2115	60708	查找算法	100	0		
28	2115	60708	查找算法	100	0		
29	2115	60708	查找算法	100	0		
30	2115	60708	查找算法	100	0		
31	2115	60708	查找算法	100	0		
32	2115	60708	查找算法	100	0		
33	2115	60708	查找算法	100	0		
34	2115	60762	查找算法	100	0		
35	2115	60762	查找算法	100	0		
36	2115	60762	查找算法	100	0		

3.选取好后，分别计算考生得分之和，提交答案人次，从第 1 次到第 4 次考试中全体考生试题得分之和的和，从第 1 次到第 4 次考试中提交答案人次之和，高分组平均成绩，低分组平均成绩，从而计算失分率、累计失分率、第 4 次考试中的区分度

```

1 k=4
2
3
4 #失分率计算
5 Xk=3916.7 #第k次测试中考生试题的得分之和
6 nk=113 #第k次测试中提交答案人次
7 SumXk=8716.77 #从第1次到第k次考试中全体考生试题得分之和
8 Sumnk=281 #从第1次到第k次考试中提交答案人次之和
9 Lk=1-(Xk/(100*nk)) #失分率
10 for i in range(1,k+1):
11     SumXk=SumXk+Xk
12     Sumnk=Sumnk+nk
13     L1k=1-(SumXk/(100*Sumnk)) #累计失分率
14     print(Lk,L1k)
15
16 #区分度计算
17
18 XHighk=92.47 #高分组平均成绩(前27%)
19 XLowk=0 #低分组平均成绩(后27%)
20 Dk = (XHighk - XLowk)/100 #第k次考试中的区分度
21 print(Dk)
22
23

```

PyDev console: starting.

Python 3.7.6 (default, Dec 30 2019, 19:38:26)
[Clang 11.0.0 (clang-1100.0.33.16)] on darwin

```

>>> runfile('/Users/huangchengdo/PycharmProjects/BigHomework/Calculate.py')
0.6533893805309734 0.6673455661664393
0.9247
>>>

```

Variables:

- Dk = {float} 0.9247
- L1k = {float} 0.6673455661664393
- Lk = {float} 0.6533893805309734
- SumXk = {float} 24383.57C... View
- Sumnk = {int} 733
- XHighk = {float} 92.47
- XLowk = {int} 0
- Xk = {float} 3916.7
- i = {int} 4
- k = {int} 4
- nk = {int} 113

4.计算查找算法一类的信度，因为计算总体方差时的 n_k 与样本方差时的 n_k 会差一个等于题目数量的倍数的值，因此需要将最后算出的总体方差乘以倍数得到真正的总体方差

case_id	user_id	case_type	final_score	score
		查找算法 总体方差		1695.50102
		排序算法 总体方差		1840.19045
		树结构 总体方差		1742.03416
		数字操作 总体方差		1556.45933
		数组 总体方差		1556.69113
		图结构 总体方差		1780.01726
		线性表 总体方差		1907.39414
		字符串 总体方差		1881.01666
		总体方差		0
		总总体方差		1785.32921

	A	B	C	D	E	F	G
1	case_id	user_id	case_type	final_score	score		
+	115	2115 方差			1533.41981		
+	241	2174 方差			1443.45806		
+	514	2397 方差			1117.34182		
+	702	2445 方差			1832.62607		
+	857	2449 方差			1628.61715		
+	1015	2450 方差			1248.40764		
+	1156	2451 方差			1438.10894		
+	1213	2453 方差			993.116883		
+	1296	2454 方差			1270.13922		
+	1420	2461 方差			651.09956		
+	1546	2462 方差			851.612903		查找算法样本方差之和
+	1636	2463 方差			1353.524		121290
+	1768	2464 方差			885.0734		
+	1852	2465 方差			1459.30062		
+	1933	2505 方差			1593.67089		
+	2118	2506 方差			1188.96412		

将总体方差乘以倍数后得到真正的总体方差：138990

The screenshot shows the PyCharm IDE with a project named 'BigHomework'. The main editor displays a file named 'reliability.py' with the following code:

```

1 #信度计算
2 t=82                                #题目数量
3 sumS2=121290                        #样本方差之和
4 sumS=138990                         #总体方差
5 Bk=(t/(t-1))*(1-(sumS2/sumS))      #信度
6 print(Bk)
7

```

The bottom panel shows the 'Python Console' with the following output:

```

Python 3.7.6 (default, Dec 30 2019, 19:38:26)
[Clang 11.0.0 (clang-1100.0.33.16)] on darwin
>>> runfile('/Users/huangchengdo/PycharmProjects/BigHomework/...')
0.12891947995192837
>>>

```

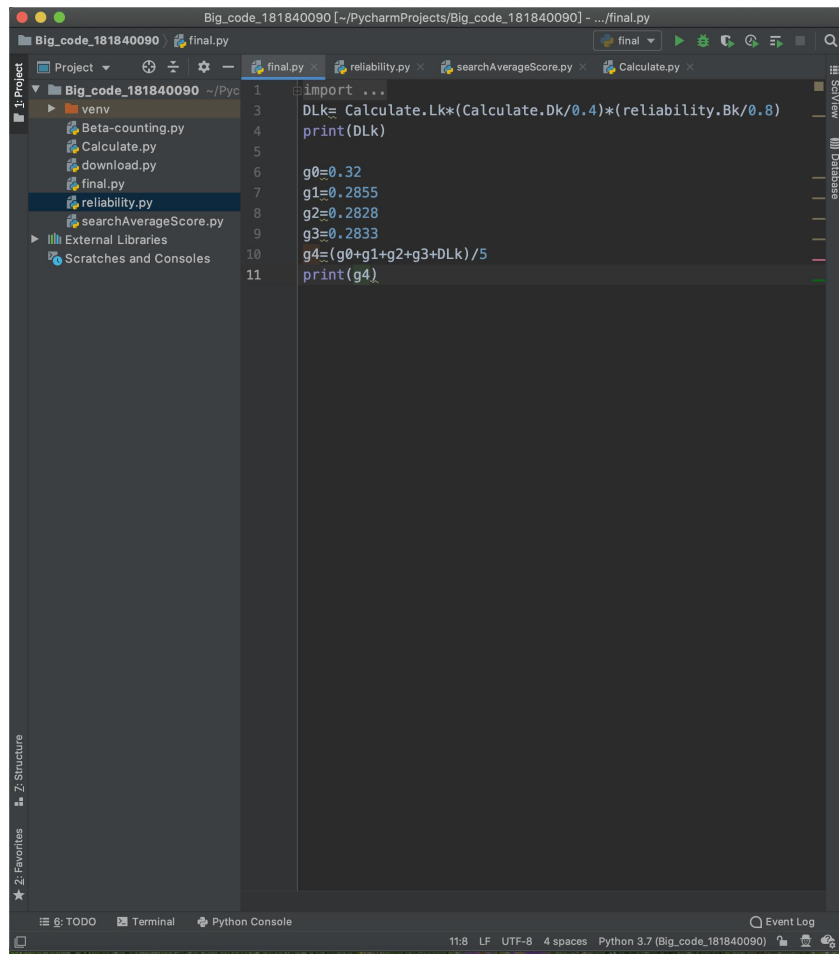
The right sidebar shows the 'Variables' window with the following values:

```

Bk = (float) 0.12891947995192837
sumS = (int) 138990
sumS2 = (int) 121290
t = (int) 82

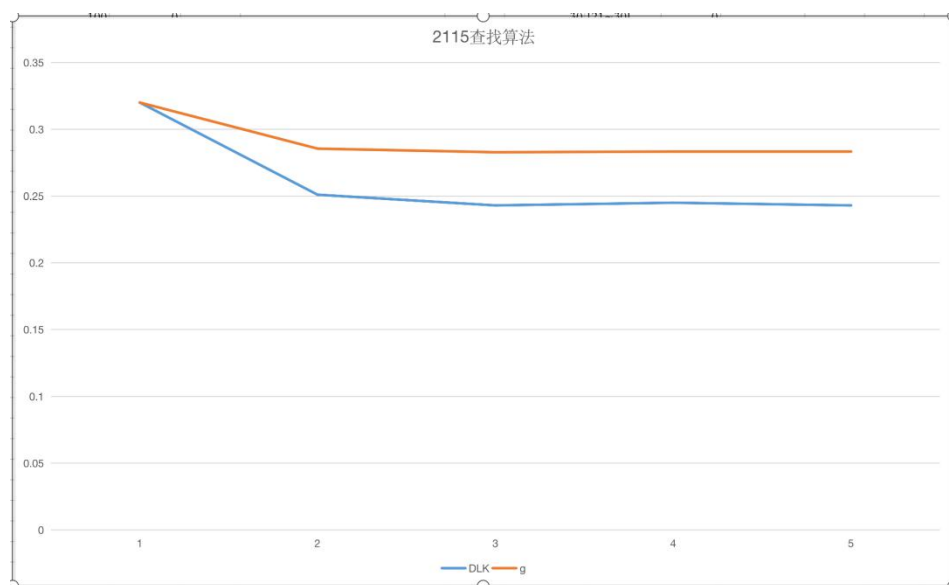
```

计算出 DLK，判断是否等于 0 或者大于等于 1，发现均为否定，因此可以直接使用，成为 DLK_4 （录屏中此处 Dk 与 Bk 参数颠倒，已经修正）



```
1 import ...
2
3 DLK= Calculate.Lk*(Calculate.Dk/0.4)*(reliability.Bk/0.8)
4 print(DLK)
5
6 g0=0.32
7 g1=0.2855
8 g2=0.2828
9 g3=0.2833
10 g4=(g0+g1+g2+g3+DLK)/5
11 print(g4)
```

在此之前，也是以方法，求出 g_1 ， g_2 ， g_3 以及对应的 DLK，得到曲线图：



可见实际上样本难度 g 快速收敛，接近于 0.28 左右

最后得到了 2115 查找算法这道平均分为 33 的题目，对应拥有 0.28 的难度系数，主要原因还是两极分化分数过于严重，而对于 2445 查找算法这道平均分 65 分的题目，则得到了 0.13 左右的难度系数，两者的信度相同，主要还是区分度导致了两者的区别，让平均分变得更加的科学，不同类型的题目也会拥有不同的信度，使得题目的难度系数很好地被定义了。

美中不足的是，信度的计算仍然是比较粗糙的，没有能够做到更加精确地分类计算，与此同时，目前对于自动化的程度仍然不足，理论上更加完美的状态应该是随着每一次提交题库都能做到动态变化，从而保证题库的有效性与准确性。

（此次研究报告从数据分析入手，选取来源于样本分析题目难度系数与统计学习等报告的思路，并对各个阶段的数据与指标选取进行了进一步详细的分析比较，并结合数据实际情况进行调整，从而得到结果）

对这门课的意见和想对老师说的话：

对这门课的意见：这门课让我初步掌握了概率论与数理统计相关内容，希望这门课可以有更多有趣的项目实验。

想对老师说的话：感谢老师一个学期的辛勤付出！希望老师这门课能越办越好！

附录：图片均在报告中，GitHub 链接中也有代码，图片和数据保存。