




分析题目提交情况对题目难度进行预测



组长：刘佳月
组员：李昉
组员：胡子华

简述

方法一

从test_data_json中获取数据后，进行数据预处理过滤不需要的数据，然后进行分析获取软件度量，最后创建相关模型进行预测题目难度。

方法二

从OJ上爬取多场比赛的提交记录，汇总数据，从而获取AC率、1A率、AC用时等有望用于表征题目难度的指标，分析相关性并创建相关模型来预测题目难度。

目录 ▶

▶ **第一部分** 方法一：软件度量

▶ **第二部分** 方法一：模型创建

▶ **第三部分** 方法二：数据获取

▶ **第四部分** 方法二：模型创建

第一部分



方法一 软件度量的获取



圈复杂度



逻辑代码行数



不同操作符数



预处理

检测代码编程语言是否是python

- 检测“#include”、“const”、“int ”、“void”等C++常见用语和“public static void main”、“System.out”等JAVA语言常见语句进行检测和判断。
- 对分号数量进行计数，超过3个判定为非python语言。

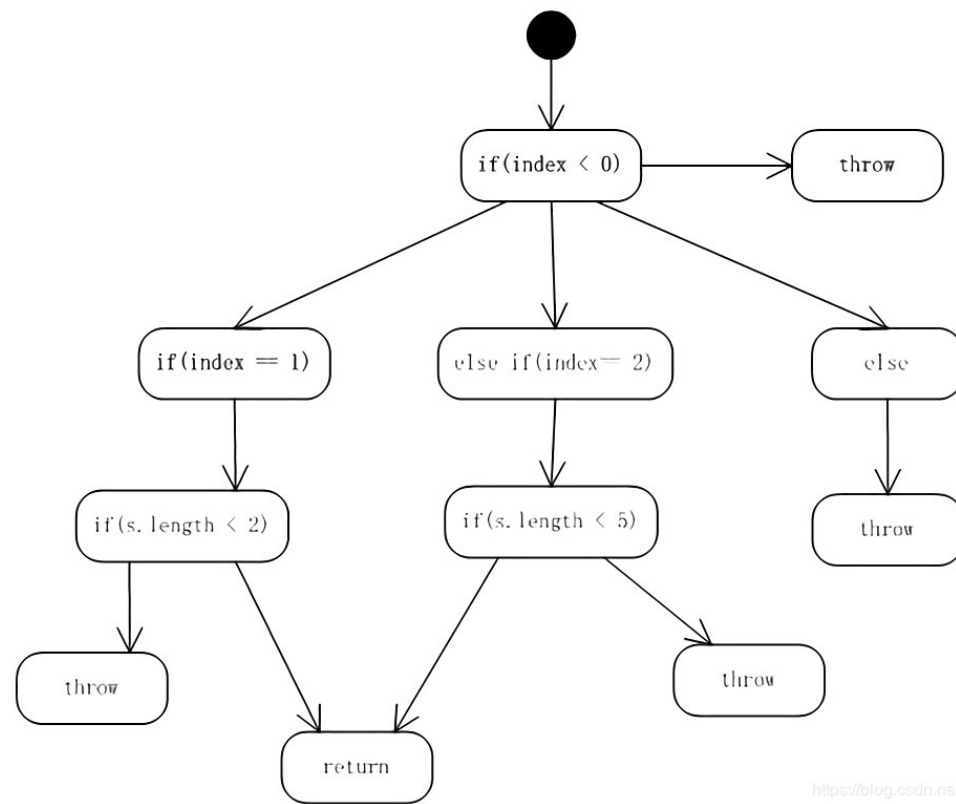
检测是否是面向测试用例编程

- print出现超过10次，判定为T0。
- suspected/line_num的比例高于阈值（ 0.3 ），判定为T0。
- print行数/总行数比例的比例高于阈值（ 0.9 ），判定为T0。
- if和print的数量之差不超过1而且 ≥ 5 设为T0。
- case和print的数量之差不超过1而且 ≥ 5 设为T0。（使用了switch case的情况）
- 所有答案都出现在代码里判定是T0。

圈复杂度

圈复杂度(Cyclomatic Complexity)是一种代码复杂度的衡量标准，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。一般而言，当一份代码中含有越多判断分支结构，其逻辑复杂程度就越高。

计算公式为： $V(G)=e-n+2$ 。其中， e 表示控制流图中边的数量， n 表示控制流图中节点的数量（包括起点和终点，所有的叶节点都只算一个节点），求出来的 $V(G)$ 即是独立现行路径条数。






逻辑代码行数

逻辑代码行数 (LLOC, logical lines of code) 是Raw Metrics的一种，指源代码经过预编译后的行数，即实际是正确逻辑的行数。



不同操作符数

不同操作符数(unique operand numbers)是Halstead Metrics的一种。操作符通常包括语言保留字、函数调用、运算符，也可以包括有关的分隔符等。它是Halstead复杂度里面的一个很重要的指标，一般来说，操作符的数量越多，程序结构就越复杂



第二部分



方法一 模型创建



数据探索



无监督学习：聚类分析

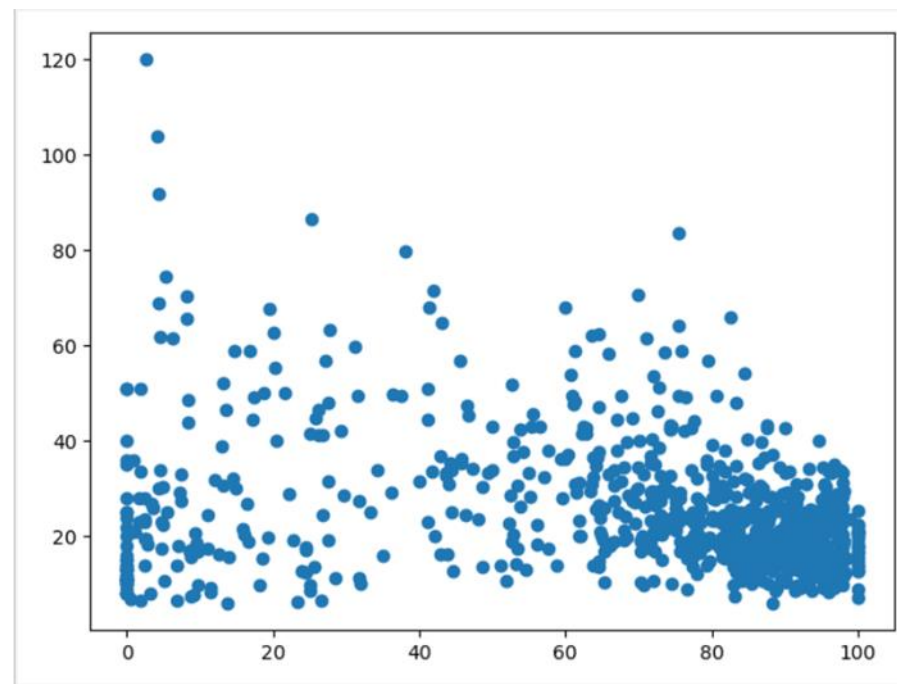


有监督学习：Adaboost分类

数据探索

分别绘制以题目平均分为X轴，三种软件度量为Y轴的散点图。

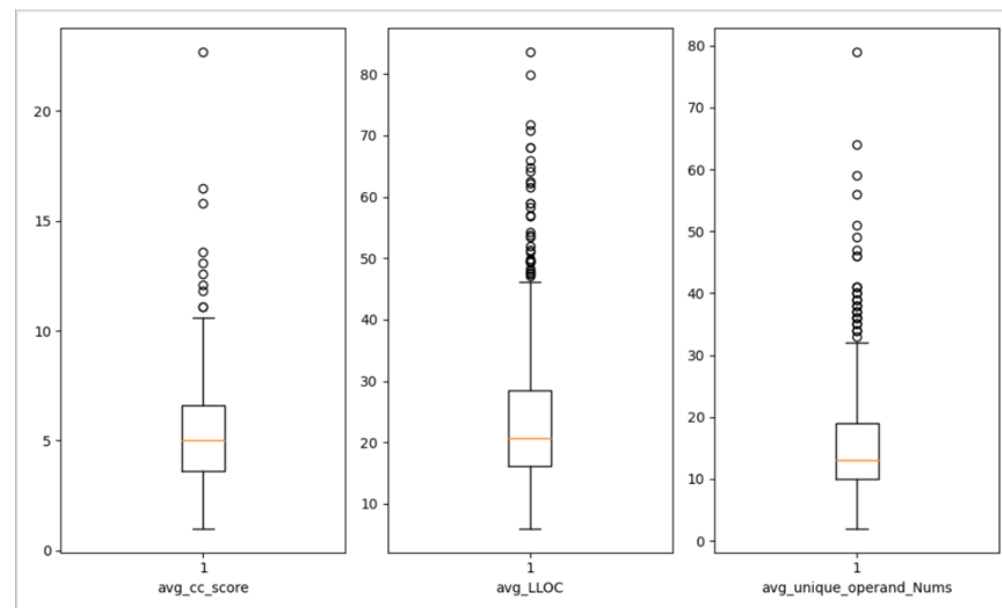
观察发现：均分在35分以下的题目，各指标的不确定性大，没有规律。可能原因是面向用例或者非python语言提交过多。因此只分析均分35-100分的题目。



无监督学习：聚类分析

- **数据清洗：**分别绘制三种度量指标的箱式图，去除离群点，以免个别异常点影响聚类结果。
- **数据归一化：**不同特征往往具有不同的量纲和单位，为了消除特征之间的量纲影响，需要进行数据标准化处理，使数据特征之间有可比性。本研究所有特征都使用离差标准化方法，对数据进行线性变换，将结果值映射到[0-1]之间。

$$\text{新数值} = \frac{\text{原数值} - \text{极小值}}{\text{极大值} - \text{极小值}}$$



圈复杂度、逻辑代码行数、不同操作符个数的箱式图

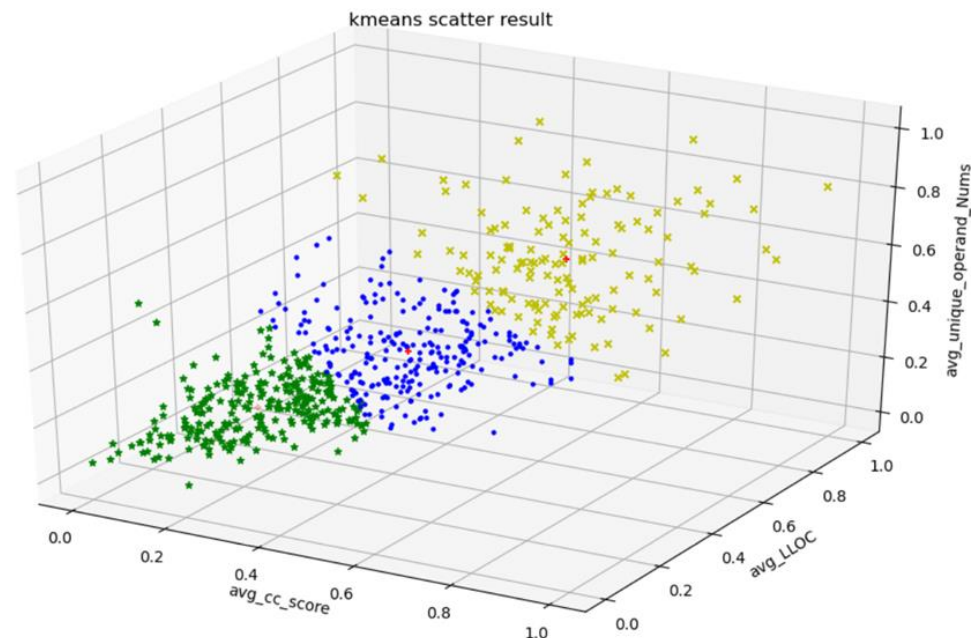
无监督学习：聚类分析

- **K-Means聚类模型：**使用K-Means聚类算法，对数据进行无监督学习。使用calinski_harabasz_score来表征分类效果，尝试调整参数n_clusters（表示类别的数量）。

Calinski-Harabasz-scores

$$s(k) = \frac{\text{tr}(B_k)}{\text{tr}(W_k)} \frac{m-k}{k-1}$$

其中m为训练集样本数，k为类别数， B_k 为类别之间的协方差矩阵， W_k 为类别内部数据的协方差矩阵，tr为矩阵的迹。通过比较，选择n_clusters为3，其聚类结果如右上图所示。



根据(逻辑代码行数、圈复杂度、不同操作符数)三维特征的聚类结果

类别	类中心	题目难度
1	(0.448,0.380,0.393)	中
2	(0.630,0.637,0.637)	难
3	(0.235,0.201,0.222)	易

聚类的质心

有监督学习：AdoBoost分类

- 获取题目难度标签

经典测试理论中，试题的难度通常用 $P=1-\frac{S}{F}$ 来确定，其中P表示试题难度，S表示被测试者在该题上得分的平均值，F表示该题的满分分数。由于数据集中的编程者不是在规定时间内中进行编程，题目的AC率、1A率、AC时长等类似指标的可参考性不大，故仅采用题目平均分来表征题目难度。将学生代码平均得分（记为x）划分为A、B、C三个等级作为实际难度指数RDI。

A: $88 \leq x < 100$ B: $60 \leq x < 88$ C: $35 \leq x < 60$

[基于“数据探索”阶段的观察，本研究只分析均分35-100分的题目]

- 训练模型

对于每一道题，使用逻辑代码行数，圈复杂度和不同操作符数量作为特征，将难度类型取值范围为{A, B, C}的题目作为机器学习的输入。使用AdoBoost集成算法，通过构造和使用多个CART弱分类器，对数据进行监督分类。AdoBoost模型首先使用GridSearchCV对框架参数n_estimators(学习器个数)进行择优，然后对CART弱学习器参数max_depth、min_sample_split进行择优。

- 最终分类准确率达到71.35%。

第三部分



 **方法二** 数据获取

 数据采集

 数据汇总

数据采集

本研究的实验数据来自英国在线评测系统Atcoder上定期举办的Grand Contest (<https://atcoder.jp/contests/archive>)的提交记录。

选择该OJ的重要原因是它为每道编程试题按照难度赋予分值，系统标记的经验难度分值可以作为本研究AdoBoost模型的输入参数和性能检测标准。同时比赛过程中的时间限制促使编程者更专心，较少受到其他因素的影响。综合考虑我们选择不存在“一题两问且分别计分”情况的比赛，便于爬取；选择提交记录页数不过多的比赛，以提高研究效率。最终我们选取了18场Grand Contest，共爬取了107道编程题的约15万条比赛期间的提交记录。

submit_time	task	user	score	status
2017-06-18 22:48:06	A - Shrinking	NB29979	0	WA
2017-06-18 22:48:04	B - Colorful Hats	domslee	700	AC
2017-06-18 22:48:03	B - Colorful Hats	zaki_	0	WA
2017-06-18 22:48:02	D - XOR Replace	pekempey	0	WA
2017-06-18 22:48:01	C - +/- Rectangle	spica314	700	AC
2017-06-18 22:47:58	C - +/- Rectangle	matonix	0	WA
2017-06-18 22:47:58	E - Poor Turkeys	dreamoon	1400	AC

某场比赛部分的提交记录

数据汇总

根据数据采集阶段爬取的各道题目的分值，我们将题目划分为A、B、C由易到难三种难度等级。将原始数据集里的提交记录按照题目编号统计汇总，统计该题的总提交次数、AC量、1A量、AC总时长、所有用户所得的总分、参与的用户数量。然后计算AC率、1A率、AC平均时长、平均得分率、提交总次数，最终得到以题目编号为关键字的训练数据集。

$$1A率 = \frac{1A量}{AC量}$$

$$平均得分率 = \frac{用户在该题的平均得分}{题目分值}$$

id	score	ac_rate	1a_rate	avg_ac_time	avg_score	score_rate	total_submit	ac_Nums	1a_Nums	ac_time	total_score	user_Nums	difficulty_level
agc003A	200	64.58	73.94	155.81	192.70	96.35	1022	660	488	102834	1320	685	A
agc003B	400	30.84	44.29	619.43	309.15	77.29	1589	490	217	303519	1960	634	A
agc003C	600	43.45	74.67	289.18	468.75	78.13	863	375	280	108442	2250	480	A
agc003D	1100	15.54	29.03	1006.31	445.75	40.52	399	62	18	62391	682	153	B
agc003E	1400	23.36	56.00	184.92	593.22	42.37	107	25	14	4623	350	59	C
agc003F	1700	55.00	36.36	207.36	1700.00	100.00	20	11	4	2281	187	11	C
agc006A	200	55.66	68.41	256.80	185.16	92.58	1166	649	444	166662	1298	701	A
agc006B	400	29.44	36.31	1030.44	300.21	75.05	1216	358	130	368897	1432	477	A
agc006C	800	9.38	46.88	303.31	195.42	24.43	341	32	15	9706	256	131	B
agc006D	1300	11.95	70.37	316.96	261.94	20.15	226	27	19	8558	351	134	C
agc006E	1500	24.73	56.52	362.17	821.43	54.76	93	23	13	8330	345	42	C
agc006F	1700	22.22	50.00	458.83	566.67	33.33	27	6	3	2753	102	18	C

处理后的数据集（部分）

第四部分



方法二

模型创建



特征提取

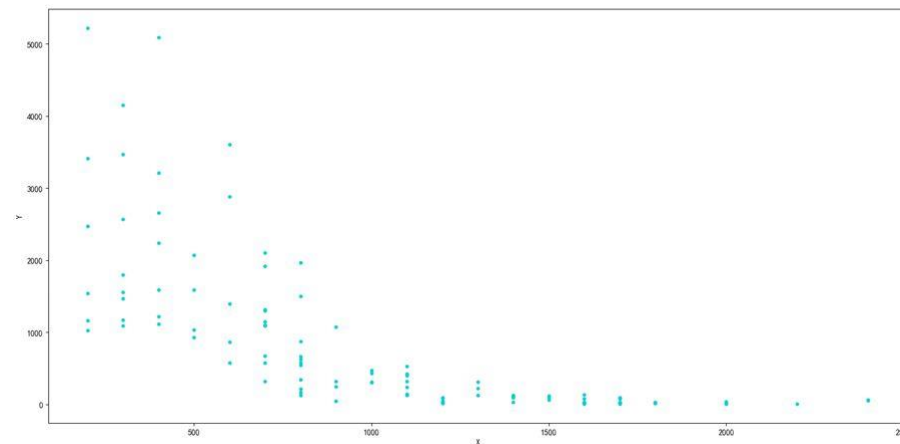


模型创建

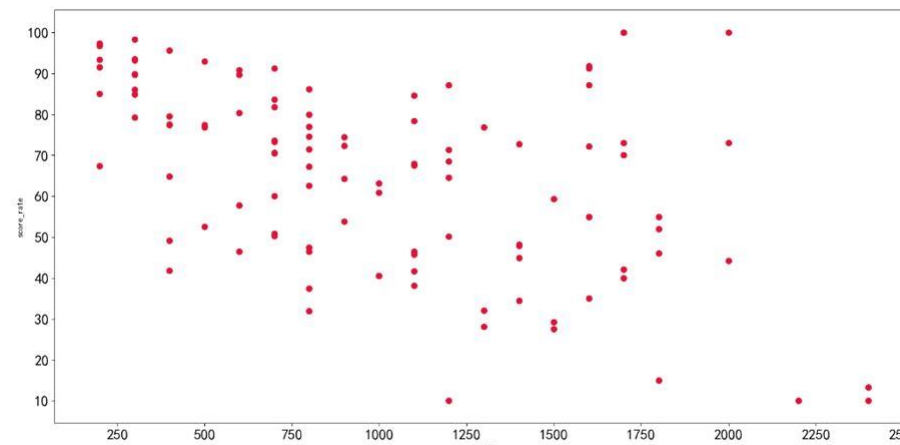
特征提取

研究之初，我们根据之前的经验认为AC率、1A率、AC平均时长在一定程度上可以衡量题目难度。接着我们采用斯皮尔曼（spearman）等级相关系数来分析提交总次数、平均得分率与题目难度的相关性。仍然根据OJ赋予题目的分值来衡量题目难度，分值越高，题目难度越大。结果得到提交总次数与题目分值的spearman系数为-0.906，平均得分率与题目分值的spearman系数为-0.461。

结合图表得出的结论，对于每一道题，经验难度类型取值范围为{A,B,C}，由数据汇总阶段划分所得；同时使用（AC率，1A率，AC总时长，平均得分率，提交总次数）多维特征来表示。



总提交次数与score难度分数呈负相关



平均得分率与score难度分数呈负相关

模型创建

本研究使用AdoBoost集成算法，通过构造和使用多个CART弱分类器，对数据进行监督分类。AdoBoost模型首先使用GridSearchCV对框架参数n_estimators（即学习器个数）进行择优，然后对CART弱学习器参数max_depth、min_sample_split进行择优。

对分类结果的测试如下表所示。

难度类型	A	B	C	整体
预测正确率	87.50%	71.43%	91.67%	85.19%