# 数据科学大作业报告

## 小组信息：

## 组员情况说明：

组长：

组员：

组员

## 研究问题：

## 代码开源地址：

# 方法一

## 1.1方法简述

## 1.2研究方法

### 1.2.1数据预处理

采用给定的数据集test\_data.json，检测异常提交，包括面向用例和非python语言提交。对于面向用例的提交，全都设置为0分；考虑到编程要求为python语言，且部分题目可以在网上找到非python语言的AC代码，对于非Python语言的提交，也全都设置为0分。最终得到每道题更真实的平均分。

#### A.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，防止部分同学小心在python代码中写上分号。

#### B.检测是否是面向测试用例编程

统计非空行、非注释的行数作为有效行数，记为n；统计"print"语句的出现次数，记为p；统计"if"、"elif"或"else"语句后面接着出现"print"语句的次数，记为s。

定义：num（A）即为A的数量

设：

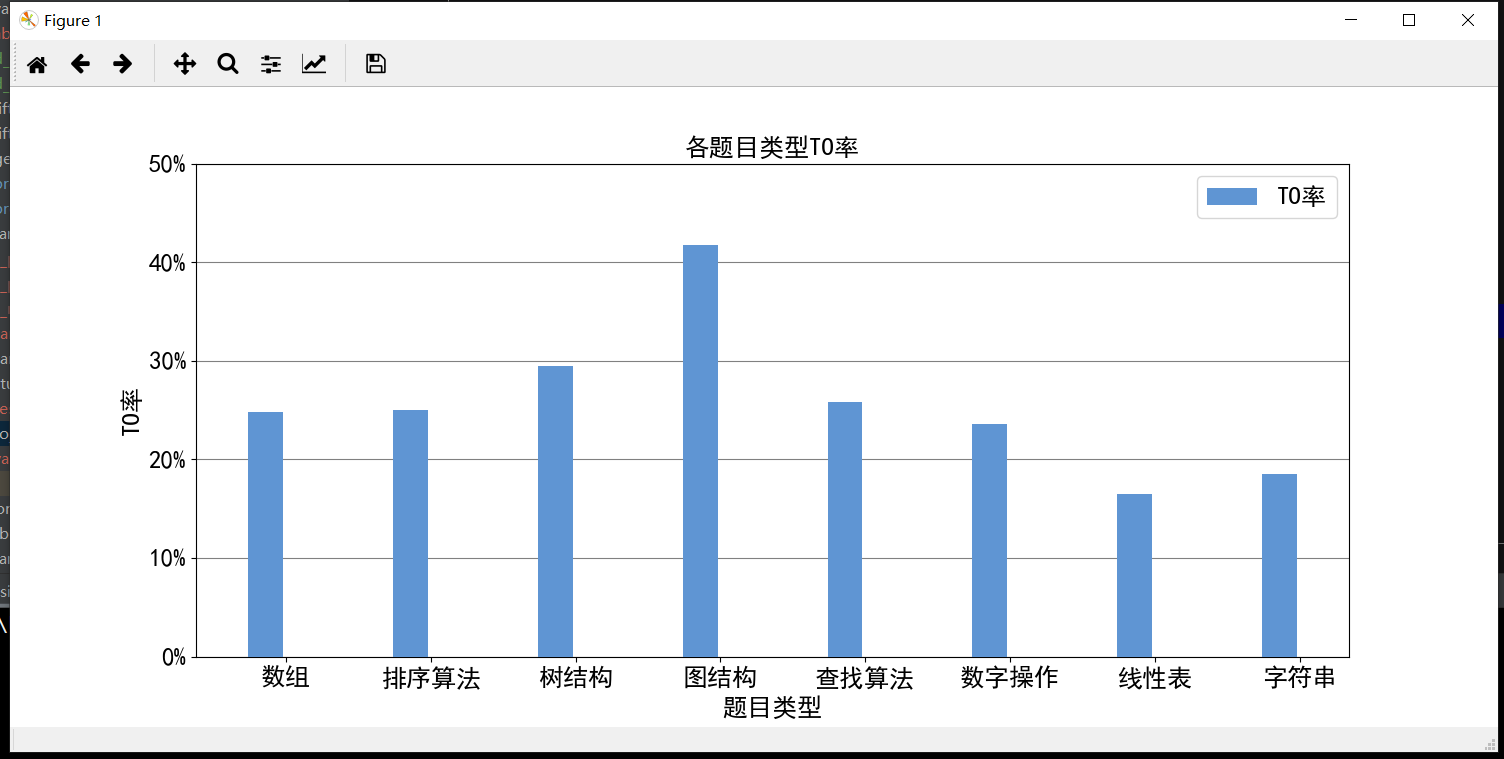
以下情况的代码判定为面向测试用例：

此外,对答案所在代码文本行进行检测，判定是不是所有该题目对应的参考答案都能一一在代码文本里找到，若是，则判断为面向测试用例编程。

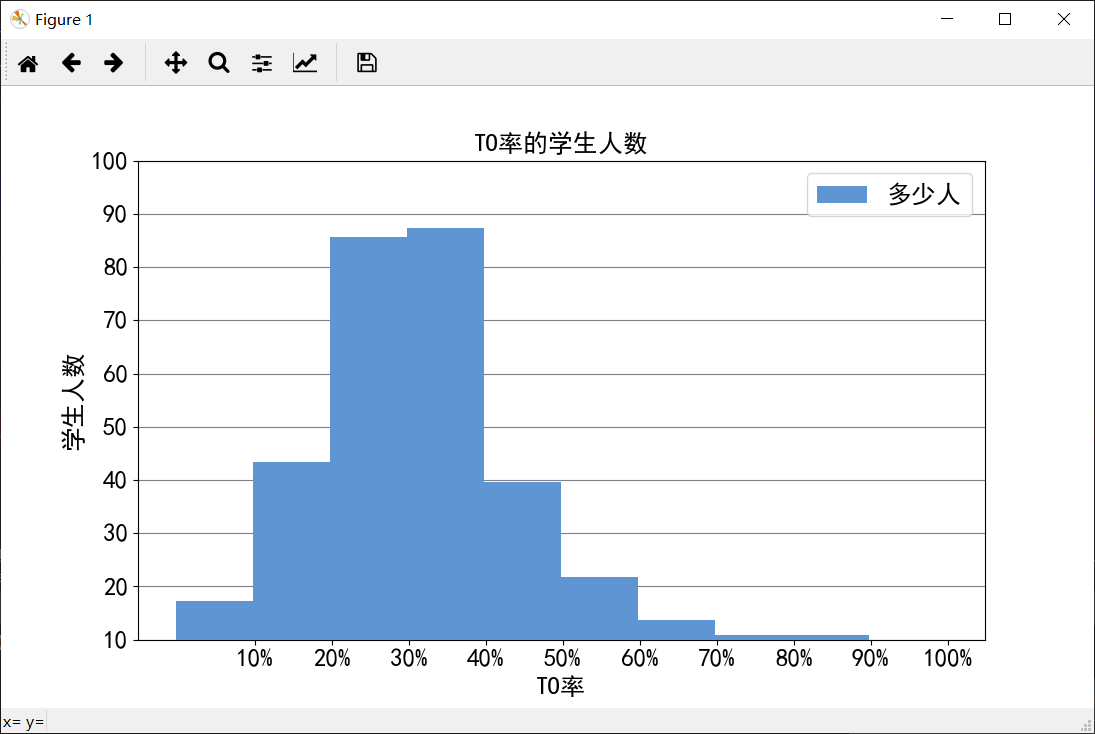
#### C.检测异常提交的准确率

以使用python语言，并且没有使用面向测试用例的样本作为有效数据集；其他一律视为异常数据集。使用random库函数random.sample()分别从有效样本和异常样本抽取50个样本，人工查看提交代码，计算检出率和误诊率。不断优化异常检测的代码，最终检出率达到？，误诊率为？。

#### D.面向对象的可视化



图表 1各题目类型面向测试用例代码比率条形图



图表 2 使用面向用例的学生的人数和比例

### 1.2.2获取软件度量（software metrics）

Radon是可以计算各种代码度量（code metrics）的python工具。选择以下的度量指标。

* 圈复杂度(Cyclomatic Complexity)

是一种代码复杂度的衡量标准，由 Thomas McCabe 于 1976年定义。它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。

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。

* 逻辑行数

是指源代码经过预编译后的行数即实际是正确逻辑的行数。

* 不同算子数

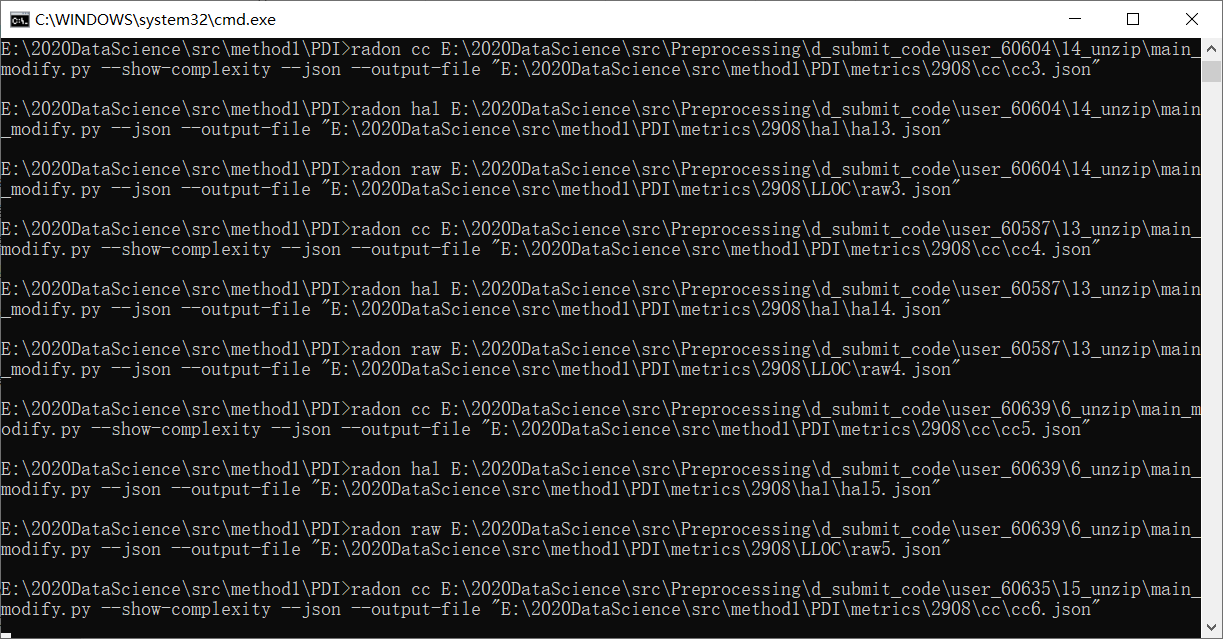
算子：广义的讲，对任何函数进行某一项操作都可以认为是一个算子，甚至包括求幂次，开方都可以认为是一个算子，只是有的算子我们用了一个符号来代替他所要进行的运算罢了，所以大家看到算子就不要纠结，它和f(x)的f没区别，它甚至和加减乘除的基本运算符号都没有区别，只是他可以对单对象操作罢了(有的符号比如大于、小于号要对多对象操作)。又比如取概率P{X<x}，概率是集合{X<x}(他是属于实数集的子集)对[0,1]区间的一个映射，我们知道实数域和[0,1]区间是可以一一映射的(这个后面再说)，所以取概率符号P，我们认为也是一个算子，和微分，积分算子算子没区别。总而言之，算子就是映射，就是关系，就是变换。

注意，radon库计算圈复杂度时，以类、函数和类中方法为单位分别计算，故需要为没有"if \_\_name\_\_==main:"的代码增加main主函数入口；由于radon库不支持中文，还需要去除代码中的注释。

做完上述处理后，对于每一道题，选择10份AC代码，分别计算度量指标并取平均，以避免个别特别精巧或者冗余的满分代码的影响，更客观地获取每道题的代码度量。

本研究使用命令行终端中调用Radon库API。先将命令写入.bat文件，双击.bat，终端开始运行所有写入的命令，批量计算每道题目的各个度量指标。

双击bat文件运行情况图：

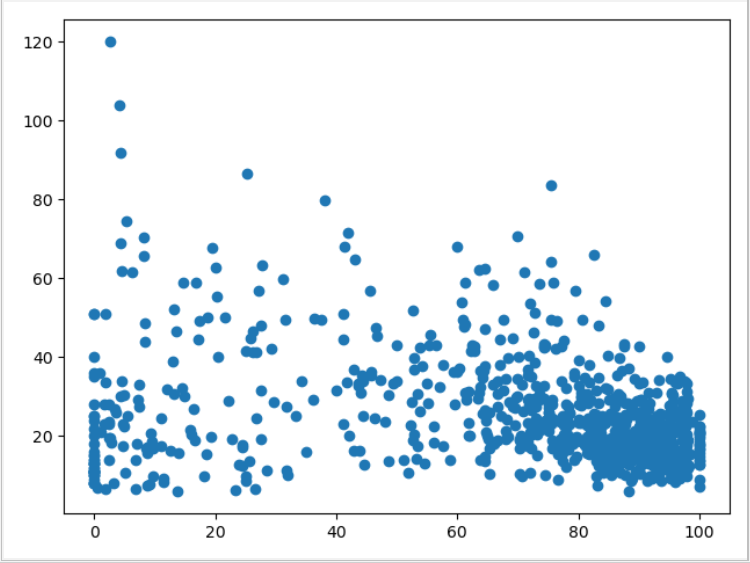


最终，得到以题目编号为关键字的训练数据集，特征属性为逻辑行数（LLOC：logical lines of code）、圈复杂度(Cyclomatic complexity)和不同算子数目(unique operand numbers)。

### 1.2.2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。因此本研究只分析均分35-100分的题目。



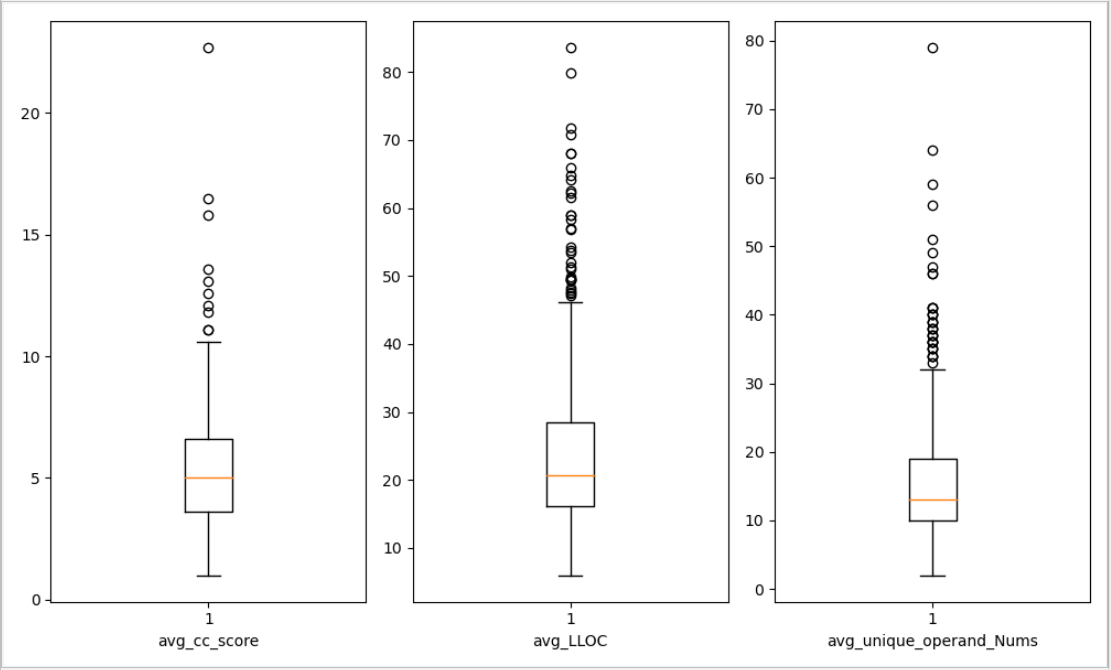
图表 3 LLOC与题目平均分的散点图

### 1.2.3创建模型

#### A.无监督学习：聚类分析

##### A.1数据清洗

由于个别异常点影响聚类结果，故分别绘制三种度量指标的箱式图，以便去除离群点。



图表 4 圈复杂度、逻辑代码行数、不同算子个数的箱式图

具有以下特征的数据点视为异常离群点

①平均圈复杂度 > 12

②平均逻辑行数 > 50

③平均特殊操作符数 > 35

##### A.2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，以解决数据特征之间的可比性。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。

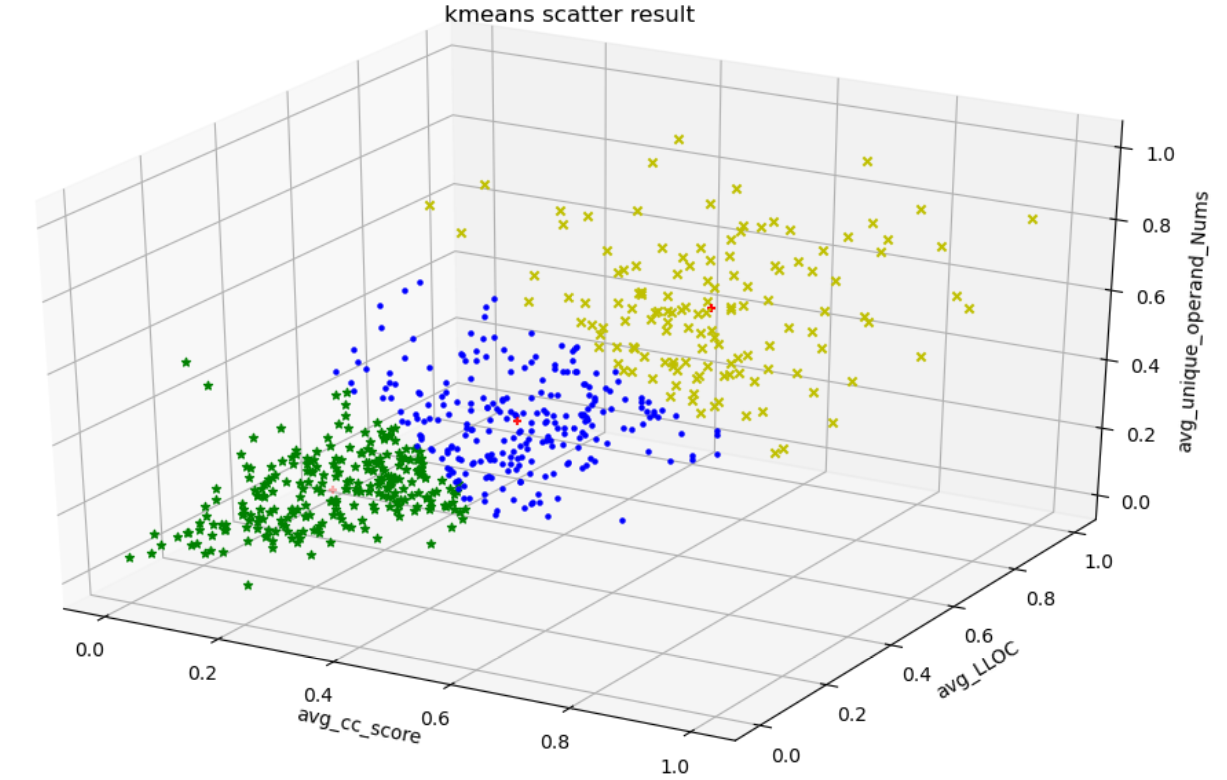


##### A.3模型创建

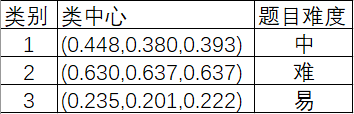
本研究使用K-Means聚类算法，对数据进行无监督学习。使用calinski\_harabasz\_score来表征分类效果，尝试调整参数n\_clusters（表示类别的数量）。

Calinski-Harabasz-scores ，其中m为训练集样本数，k为类别数，Bk为类别之间的协方差矩阵，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，其聚类结果如下图所示。



图表 5 根据（圈复杂度、逻辑代码行数、不同算子数量）三维特征的聚类结果



表格 1 聚类的质心

#### B.有监督学习：AdoBoost分类

##### B.1获取题目难度标签

经典测试理论中，试题的难度通常用来确定，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。此外，由于数据集中的编程者不是在规定时间中进行编程，题目的AC率、1A率、AC时长等类似指标的可参考性不大。故使用本数据集进行有监督学习时，仅采用题目平均分来表征题目难度。

将学生代码的平均得分（记为x）划分为A、B、C、D四个等级作为实际难度指数RDI

A ：88<=x<100

B ：60<=x<88

C ：35<=x<60

D ：x<35

##### B.2训练模型

对于每一道题，使用（逻辑代码行数，圈复杂度，不同算子数量）三维特征来表示。由于“1.2.2数据探索”阶段的观察，本研究只分析均分35-100分的题目，故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（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。

最终分类准确率达到71.35%。

## 1.3案例分析

# 方法二

## 2.1方法简述

## 2.2研究方法

### 2.2.1所采用的数据集

本研究的实验数据来自英国在线评测系统Atcoder（https://atcoder.jp/）。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。为了使采集的样本更客观地反映题目难度，我们选择爬取Atcoder上定期举办的Grand Contest（https://atcoder.jp/contests/archive）的提交记录，因为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。

### 2.2.2数据采集

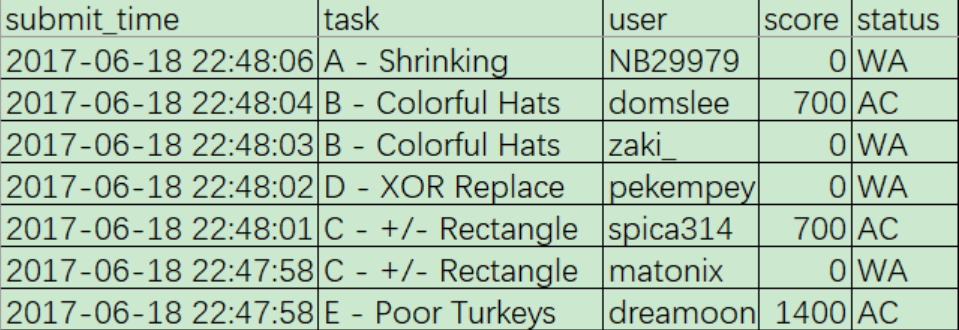
首先爬取从2016年7月至今AtCoder定期举办的共45场Grand Contest的基本情况，包括题量、每题所赋予的分值、提交记录页数。可以得知：每场比赛基本都是6题，但也有比赛存在一题两问且分别计分的情况；提交记录少则300多页（约6000条提交记录），多则高达900多页（近20万条提交记录）。

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。

### 2.2.3数据汇总

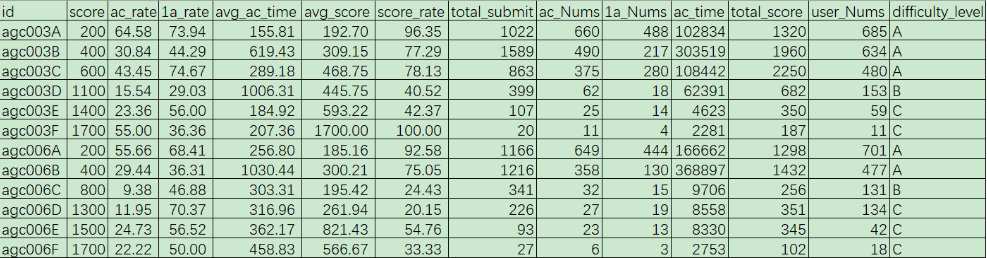
根据数据采集阶段爬取的各道题目的分值，我们将题目划分为A、B、C由易到难三种难度等级。

同时爬取的原始数据集为各场比赛用户的提交记录，包含的特征属性有提交时刻、题目名称、提交用户、运行结果、所得分数。



表格 2 某场比赛部分的提交记录

我们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。其中，，。



表格 3处理后的数据集（部分）

### 2.2.4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。



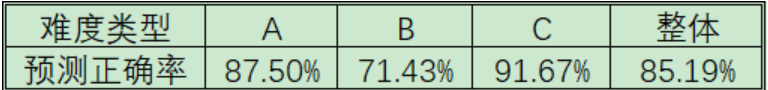
缺两幅散点图

综合以上，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）五维特征来表示。

### 2.2.5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。



表格 4

## 2.3案例分析