

2020

数据科学大作业

Data Science
By 刘一铭 檀潮 王宇博



目录

COMPANY

01

项目简介

What we have done

02

代码介绍

Presentation of our code

03

数据成果展示

The result of the project

04

PCA


The work for dimension reduction



第一部分

项目简介

What we have done
A brief introduction of the whole project



项目简介

A brief introduction of the whole project



提出思路

给出最初的项目目标
提出分3个阶段逐步完成作业项目



第一阶段

提出掌握值的概念和计算指标
按四个指标分别检验数据有效性



第二阶段

计算掌握值
八种题型下分别验证掌握值分布



第三阶段

对原始数据进行处理得到新数据
实现对学生个性化题目推荐功能

项目简介

A brief introduction of the whole project



提出思路

我们小组最开始希望能够通过提供的数据实现针对每个学生的个性化题目推荐，能够依据该学生目前已经提交的题目记录找出目前最适合该学生的题目。

起初项目中涉及对题目进行进一步分组，提出我们自己的标签，依据标签相似程度推荐题目，后期则改变实现思路，通过验证已有八种类型的合理性，在已有类型上进行推荐。


最终提出将项目分为三段，第一先要给出一种衡量学生编程能力的指标，其次在该指标基础上根据学生得分验证已有题目分类是否合理，最后根据学生数据对其推荐题目。



第二部分

代码介绍

Presentation of our code
Including the code to implement the function



代码介绍

Presentation of our code



衡量指标

提出四种指标用于衡量学生的编码水平
并对四种指标有效性进行验证



分类验证

将第一阶段四指标整合得到学生掌握值
根据掌握值得分对分类进行合理性验证



题目推荐

对于学生已有数据进行个性化题目推荐



代码介绍

Presentation of our code

第一阶段

- ◆ 对原有数据进行调整，得到新格式的数据
- ◆ 四个指标的计算代码（以program_rate为例）
- ◆ 分别为program_rate, debug_rate, early_success_degree, finish_degree

```
classified_test = {"排序算法": {}, "数字操作": {}, "数组": {}, "树结构": {}, "图结构": {}, "查找算法": {}, "字符串": {}}
case_test = {}

student_num = 0
case_num = 0
for value in data.values():
    student_num += 1
    user_id = value["user_id"]
    for case in value["cases"]:
        case_num += 1
        cgry = classified_test[case["case_type"]]
        if cgry.get(case["case_id"], 0) == 0:
            cgry[case["case_id"]] = []
        if case_test.get(case["case_id"], 0) == 0:
            case_test[case["case_id"]] = []

        new_case = {}
        new_case["user_id"] = user_id
        new_case["case_id"] = case["case_id"]
        new_case["case_type"] = case["case_type"]
        new_case["case_zip"] = case["case_zip"]
        new_case["final_score"] = case["final_score"]
        new_case["upload_records"] = case["upload_records"]
        cgry[case["case_id"]].append(new_case)
        case_test[case["case_id"]].append(new_case)
    print("student_num = " + str(student_num))
    print("case_num = " + str(case_num))

key_num = 0
for key in classified_test.keys():
    key_num += 1
    print("key_num = " + str(key_num))
    print()
```

```
# program_rate
# 理由：攻克题目速度越快越好
for caseid in data.keys():
    cases = data[caseid]
    for case in cases:
        records = case["upload_records"]
        if len(records) >= 4:
            time_span = records[-1]["upload_time"] - records[0]["upload_time"]
            to_cal = (max([i["score"] for i in records]) / 100) / (
                time_span / 1000 / 60 / 60) if time_span != 0 else -1 # 只考虑大于0的数据，单位：h-1
            data_to_process[caseid][case["user_id"]]["program_rate"] = math.log(to_cal + 1,
                                                                              math.e) if to_cal > 0 else INVALID
        if max([i["score"] for i in records]) == 0:
            data_to_process[caseid][case["user_id"]]["program_rate"] = 0
    else:
        data_to_process[caseid][case["user_id"]]["program_rate"] = INVALID
```


代码介绍

Presentation of our code

第二阶段

- ◆ 将四个指标综合计算掌握值
- ◆ 得到的数据进一步封装成新文件供下一步使用

```
score_data = {"program_rate": [], "debug_rate": [], "early_success_degree": [], "finish_degree": []}
for userId in data.keys():
    cases = data[userId]['cases']
    for case in cases:
        tmp_score = {"program_rate": getPro(case), "debug_rate": getDebug(case), "early_success_degree": 
            "finish_degree": getFinish(case)}
        for value in tmp_score.keys():
            if tmp_score[value] != INVALID:
                score_data[value].append(tmp_score[value])
position_data = {}
for value in score_data.keys():
    tmp_list = score_data[value]
    avg = sum(tmp_list) / len(tmp_list) if len(tmp_list) > 0 else INVALID
    var = math.sqrt(
        sum(map(lambda x: (x - avg) * (x - avg), tmp_list)) / (len(tmp_list) - 1)) # 因为取指标时需要满足严
    position_data[value] = (avg, var)
print("各指标的均值与方差, 格式: (avg,var), ", position_data)
print()

# 为每个同学计算掌握值, 无效的指标用平均值代替
# dataToWrite = []
```

```
# 对masterValuesForCase进一步按照题型分类, 每个题型下对应不同学生在该题型上的数据
classifyByType = {'str': [], 'line': [], 'arr': [], 'find': [], 'num': [], 'sort': [], 'tree': [], 'gra': []}
for userId in dataToWrite.keys():
    dataOfThisStudent = dataToWrite[userId]
    for caseType in dataOfThisStudent.keys():
        dataOfThisType = dataOfThisStudent[caseType]
        if len(dataOfThisType) > 0:
            classifyByType[caseType].append(dataOfThisType)
with open("data/masterValue.json", "w", encoding='utf-8') as f:
    json.dump(classifyByType, f, ensure_ascii=False, indent=4)
```

代码介绍

Presentation of our code

第三阶段

- ◆ 根据学生数据及前两个阶段所得结果对学生进行题目推荐


```
def getTest(cases, type, userScore, positionData):
    oldTests = []
    newTests = []
    caseIds = set([i["case_id"] for i in cases])
    classifiedData = json.load(open('../Section 1/data/classified_data.json', 'r', encoding='utf-8'))
    all_cases = classifiedData[type]
    testIdList = set(all_cases.keys()) - caseIds
    if testIdList: # 对没做过的题目进行排序
        avgUserScores = {} # 键值对为(testId:score)
        for testId in testIdList:
            userCases = all_cases[testId]
            scores = []
            tmpUserIds = []
            for case in userCases:
                if case["user_id"] not in tmpUserIds: # 可能有重复
                    tmpUserIds.append(case["user_id"])
                    scores.append(getScore(case, positionData))
            avgUserScores[testId] = sum(scores) / len(scores) if scores else 0
        newTests = sorted(avgUserScores.keys(),
                           key=lambda x: abs(avgUserScores[x] - userScore)) # 未做过的题目按平均掌握值与用户掌
    print("userScore (用户在该题型上的掌握值):", userScore)
    print("testIds not tried",
          sorted(avgUserScores.items(), key=lambda x: abs(x[1] - userScore)))
```



第三部分

数据成果展示

The result of the project



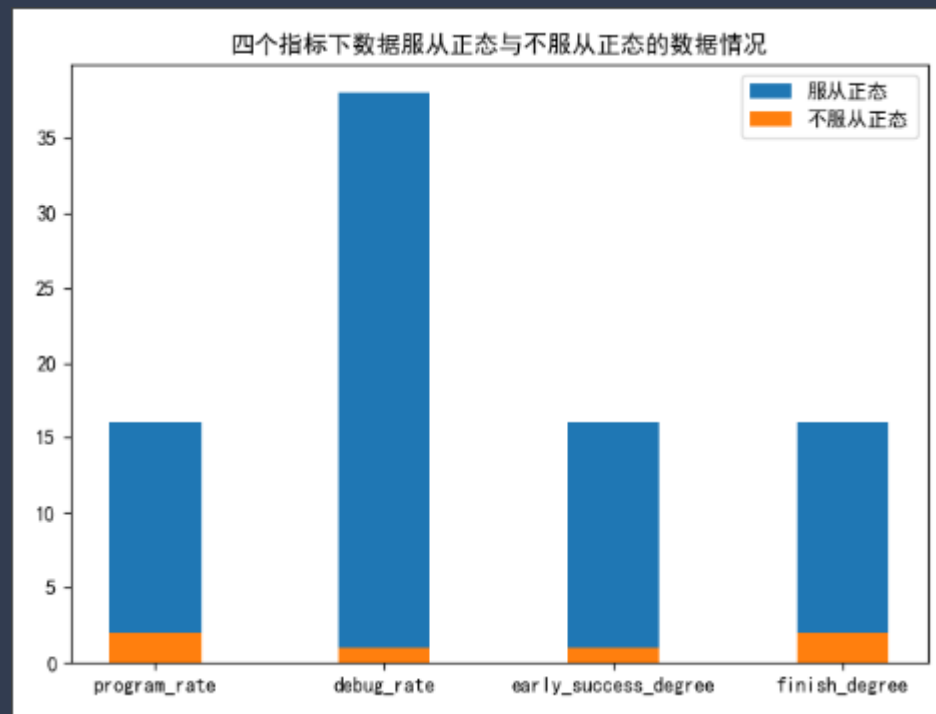
正态性检验

Normality test

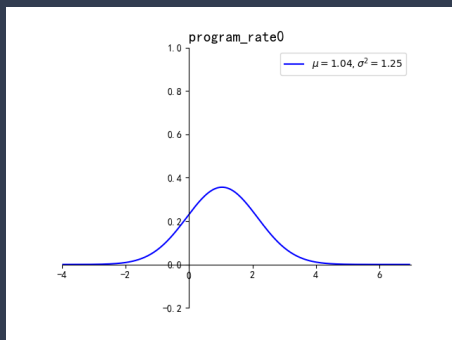
Section 1

对四个度量指标下的数据进行正态性检验

- ◆ 采用了偏度、峰度检验法
- ◆ 检验结果（各指标下数据符合正态的比例）：
 - program_rate: 87.50%
 - debug_rate: 97.37%
 - early_success_degree: 93.75%
 - finish_degree: 87.50%
- ◆ 上述四个度量指标下的数据服从正态分布的比例均在85%以上，能够有效区分学生的水平，由此我们认为所选的度量指标较为合理，可以作为刻画学生对于题目掌握程度的度量标准

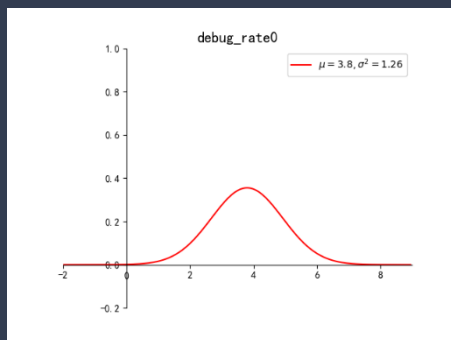


四个指标下数据正态曲线 (各取一组数据为例)



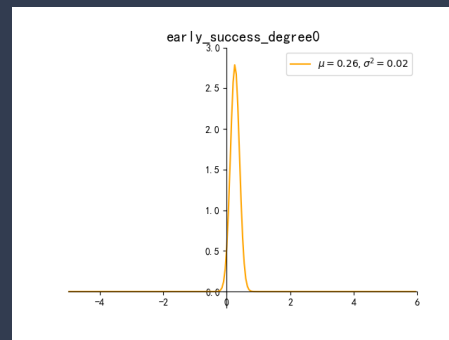
program_rate

$\mu = 1.04, \sigma^2 = 1.25$



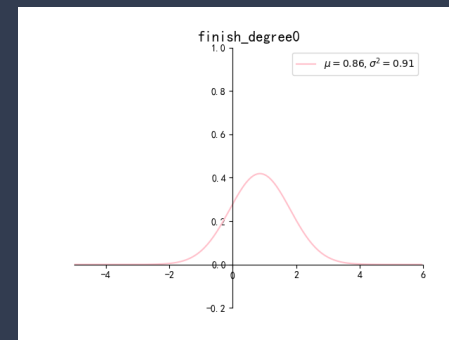
debug_rate

$\mu = 3.8, \sigma^2 = 1.26$



early_success_degree

$\mu = 0.26, \sigma^2 = 0.02$



finish_degree

$\mu = 0.86, \sigma^2 = 0.91$

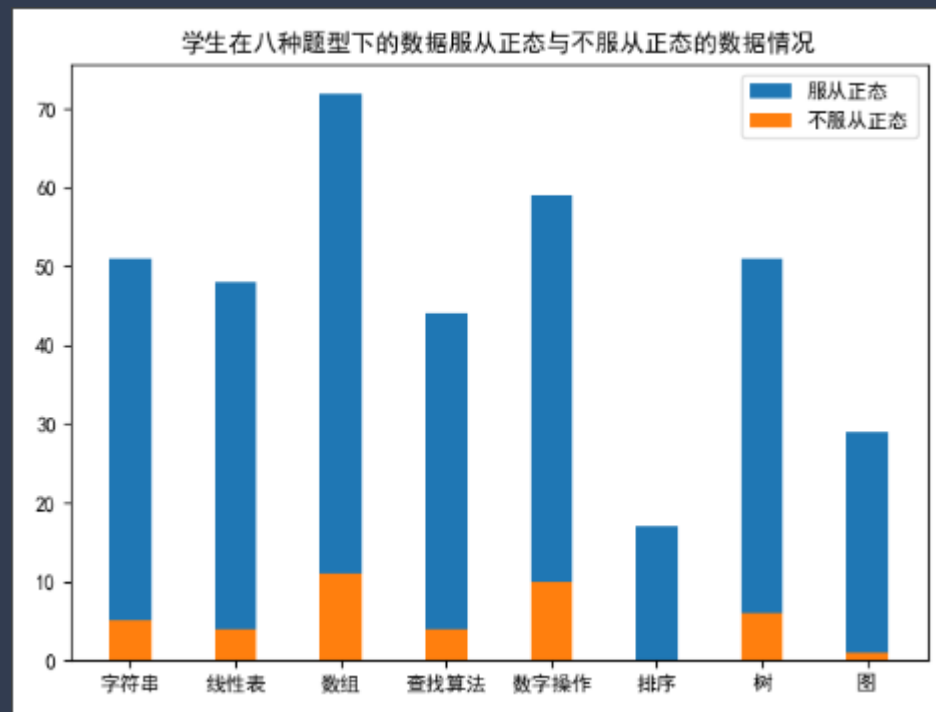
正态性检验

Normality test

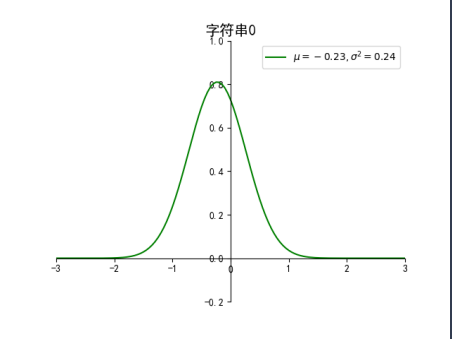
Section 2

对八种题型下的数据进行正态性检验

- ◆ 采用了偏度、峰度检验法
- ◆ 检验结果（各指标下数据符合正态的比例）：
 - 字符串：90.19% 数字操作：83.05%
 - 线性表：91.67% 排序：100%
 - 数组：84.72% 树：88.24%
 - 查找算法：90.91% 图：96.55%
- ◆ 八种题型下掌握值数据服从正态分布的学生比例均在80%以上，平均在90%以上。换言之，题目难度分布使得每个学生的在同一题型下的各题目分数分布近似符合正态。由此我们认为各题型下题目的难度设置较为合理

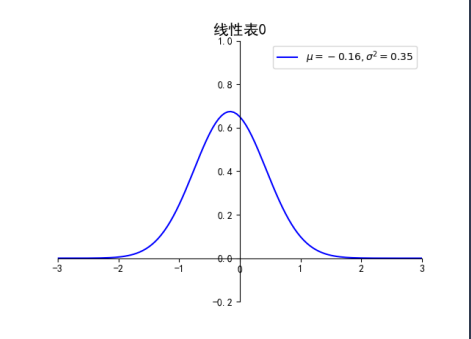


八种题型下数据正态曲线 (各取一组数据为例)



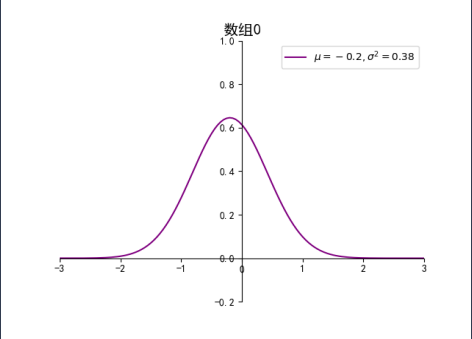
字符串

$\mu = -0.23, \sigma^2 = 0.24$



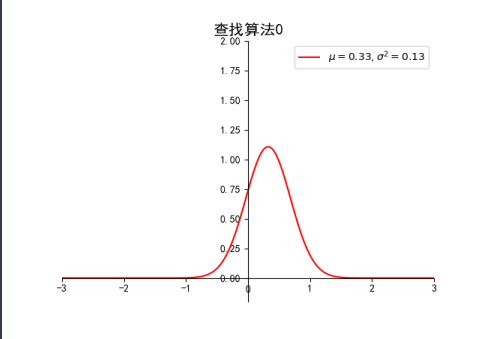
线性表

$\mu = -0.16, \sigma^2 = 0.35$



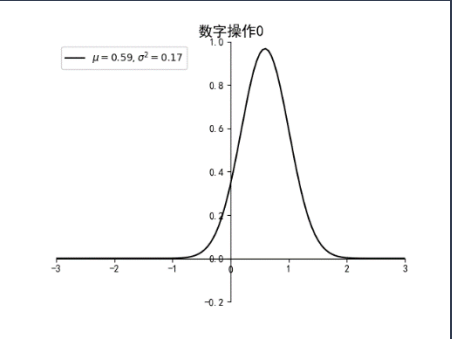
数组

$\mu = -0.2, \sigma^2 = 0.38$



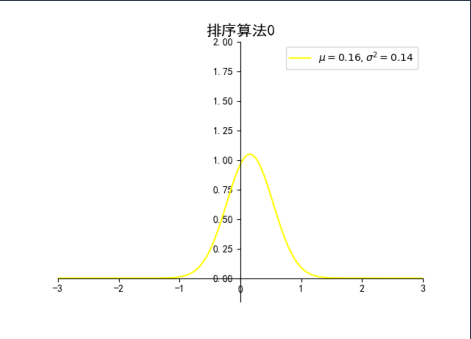
查找算法

$\mu = 0.33, \sigma^2 = 0.13$



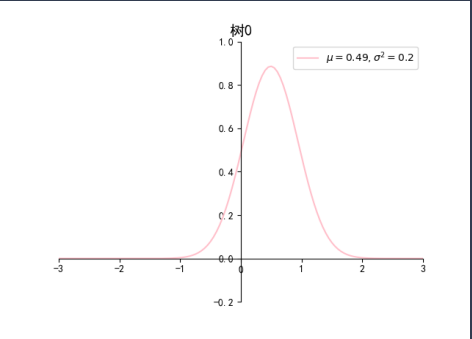
数字操作

$\mu = 0.59, \sigma^2 = 0.17$



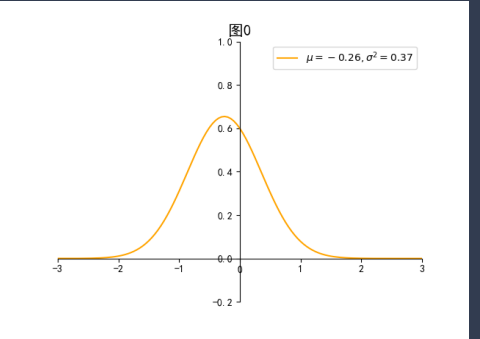
排序

$\mu = 0.16, \sigma^2 = 0.14$



树

$\mu = 0.49, \sigma^2 = 0.2$



图

$\mu = -0.26, \sigma^2 = 0.37$



第四部分

PCA

The work of dimension reduction



主成分分析 (PCA) 降维

1

数据初步处理

首先将之前基于每道题的掌握值数据进行处理，以某位学生的数据为例，将其在一种题型下的题目上的掌握值的均值作为其对该题型的掌握值，生成初步的数据集

数据筛选

留下八种题型的掌握值均存在的学生的数据，形成了PCA降维所需的数据集

2

3

降维

通过调用Python中的sklearn.decomposition库中的PCA模块，我们尝试保留源数据信息的80%进行降维，最终将八维数据降为了五维

发现

从投影后各特征维度的方差比例可以看出，前三个主成分分别占了40.7%、17.3%、11.7%的方差比例，可以说该数据集的特征还是较为分散，并没有特别集中

4

谢 谢