

数据挖掘设计报告

**ID3分类算法**

|  |  |  |  |
| --- | --- | --- | --- |
| **指标** | **评分细则** | **分值** | 得分 |
| 实现难度 | 1.算法步骤繁琐、理解困难。 | 30 |  |
| 界面 | 1.界面简洁，友好。  2.能选择输入数据 | 10 |  |
| 边界 | 1.算法边界清楚。 | 10 |  |
| 技术含量 | 1.软件的编写过程过中使用了较好的编程技巧。 | 10 |  |
| 论文质量 | 1.条理清楚、语言准确、图标清晰、格式正确 | 10 |  |
| 运行效果 | 1.算法基本能正常运行并且运行无错误。 | 30 |  |
| 总分 |  | 100 |  |

|  |  |
| --- | --- |
| 专业名称 | 专业名称 |
| 班级学号 | 00 |
| 学生姓名 | **姓名** |
| 指导老师 | 赵琪珲 |
| 提交时间 | 2000年0月0日 |

目录

[1 背景简介 3](#_Toc164679073)

[2 算法应用 4](#_Toc164679074)

[2.1 数据预处理 4](#_Toc164679075)

[2.2 算法流程图 7](#_Toc164679076)

[2.3 算法伪代码 8](#_Toc164679077)

[2.4 程序说明 9](#_Toc164679078)

[2.5 时间和空间复杂度 11](#_Toc164679079)

[2.6 算法边界 11](#_Toc164679080)

[3 运行结果 12](#_Toc164679081)

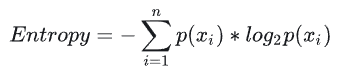
[4 结论 22](#_Toc164679082)

[5 附录 24](#_Toc164679083)

## 1 背景简介

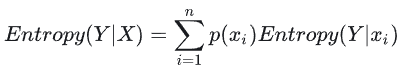
ID3算法是决策树模型算法，它是一种基本的分类回归方法，通常运用于大数据挖掘方面。ID3算法最早由罗斯昆（J. Ross Quinlan）于1975年在悉尼大学提出，算法的核心是“信息熵”。ID3算法通过计算每个属性的信息增益，认为信息增益高的是好属性，每次划分选取信息增益最高的属性为划分标准，重复这个过程，直至生成一个能完美分类训练样例的决策树。

信息熵表示信息源各可能事件发生的不确定性。其计算公式为：



其中，p(xi)是分类 xi出现的概率，n是分类的数目。

对于在X的条件下Y的条件熵，是指在X的信息之后，Y这个变量的信息量（不确定性）的大小，计算公式如下：



信息增益是在划分数据集之前之后信息发生的变化，计算每个特征值划分数据集获得的信息增益，获得信息增益最高的特征就是最好的选择。定义属性A对数据集D的信息增益为infoGain(D|A)，其计算公式为：



ID3算法的步骤为：

1. 从根节点开始，计算所有可能的特征的信息增益，选择信息增益最大的特征作为节点的划分特征；
2. 由该特征的不同取值建立子节点；
3. 对子节点递归调用本算法，构建决策树；
4. 直到没有特征可以选择或类别完全相同为止，得到最终的决策树。

## 2 算法应用

本章首先介绍算法具体应用过程，包括数据预处理、算法流程图、算法伪代码、程序说明，然后分析它的时间和空间复杂度，最后说明算法的边界。

### 2.1 数据预处理

本算法采用kaggle公开数据集Banking Dataset Classification中的train.csv，元数据基本情况如下：

数据来自于一家葡萄牙银行的电话营销活动，银行的客户将被多次联系确认是否要订购银行的定期存款产品。train.csv中包含32950条数据，每条数据包括16个属性：

Age：整数，表示年龄。

job：枚举，表示职业。包括 ('admin','blue-collar','entrepreneur','housemaid','management','retired','self-employed','services','student','technician','unemployed','unknown')

marital：枚举，表示婚姻状况。包括： ('divorced','married','single','unknown'；注: 'divorced' 包括离婚与丧偶)

education：枚举，表示受教育程度。包括：('basic.4y','basic.6y','basic.9y','high.school','illiterate','professional.course','university.degree','unknown')

default：枚举，表示信用违约状况。包括：('no','yes','unknown')

housing：枚举，表示住房贷款状况。包括：('no','yes','unknown')

loan：枚举，表示个人贷款状况。包括：('no','yes','unknown')

contact：枚举，表示常用联系方式。包括：('cellular','telephone')

month：枚举，表示最后一次联系的月份。包括：（'jan'，'feb'，'mar'，…，'nov'，'dec'）

day\_of\_week：枚举，表示最后一次联系是星期几。包括('mon','tue','wed','thu','fri')

duration：整数，表示最后一次联系时电话的持续时长，以秒为单位。

campaign：整数，在本次营销活动中联系该客户的次数（包括最后一次联系）

pdays：整数，自上次营销活动最后一次联系客户以来经过的天数（999表示以前未联系过客户）

previous：整数，在本次营销活动之前联系该客户的次数。

poutcome：枚举，上次营销活动的结果。包括('failure','nonexistent','success')

y:枚举，该客户是否在本次营销活动中订购了产品。包括('no','yes')

为便于计算，对数据做如下处理：

'job', 'marital', 'education', 'default', 'housing', 'loan', 'contact', 'month', 'day\_of\_week', 'poutcome', 'y'共11条枚举属性根据各自枚举内容建立映射关系，将枚举类型转换为整数类型。如下图所示：

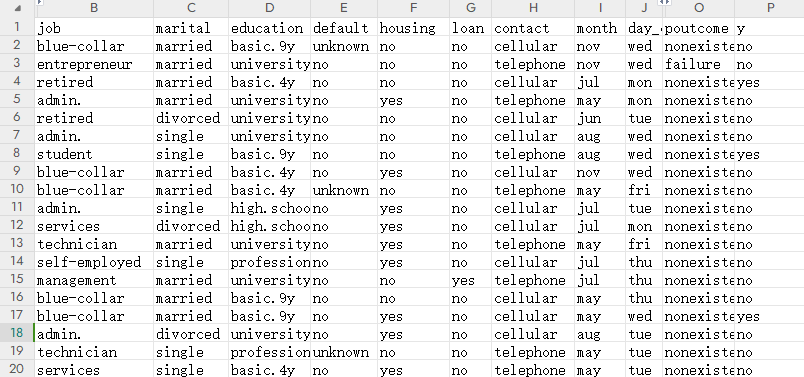


图 1 枚举类型数据转换前

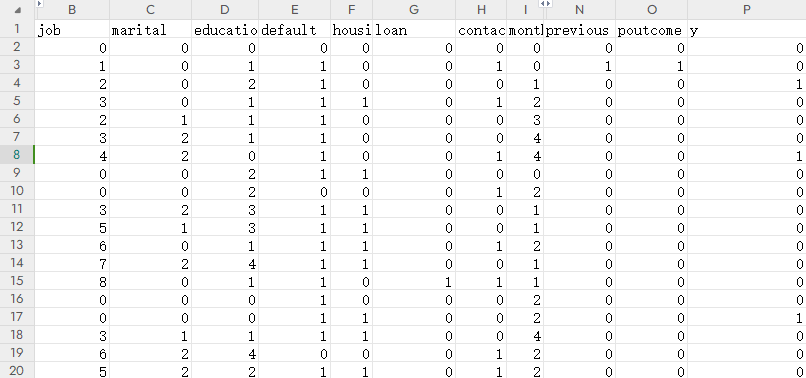


图 2 枚举类型数据转换后

'campaign','pdays','duration','age'共4条整数类型属性因取值过多，做如下处理：

'campaign'介于1到5之间时不做处理，介于6-10之间时设为6，介于11-20之间时设为7，大于20设为8。

'pdays'为999时设为0，其余数值做x/7+1操作，即按照一星期为间隔划分。

'duration'大于1200（20分钟）时设为5，其余数值做x/240操作，即按照4分钟为间隔划分。

'age'做x/10操作，即按照十岁为间隔划分。

处理完成后如下图所示：

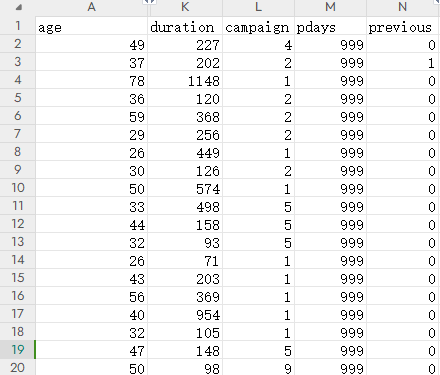


图 3 数字类型数据转换前

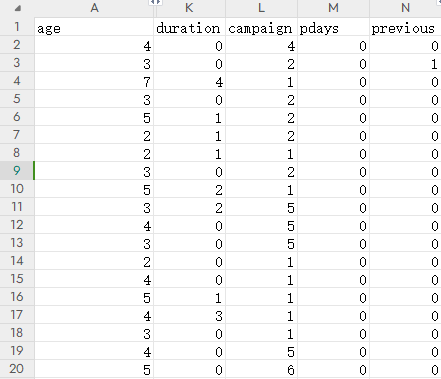


图 4 数字类型数据转换后

### 2.2 算法流程图

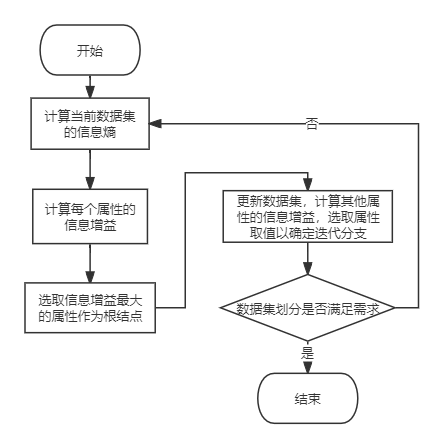


图 5 算法流程图

### 2.3 算法伪代码

预先处理如下：

|  |
| --- |
| # 读取数据  # 获取数据数量，提取目标属性列 |

主函数如下：

|  |
| --- |
| def main():  # 计算信息熵  # 计算每个属性的信息增益  # 对信息增益排序，取信息增益最大的属性，比较取值情况，选择有价值的方向进行下一轮  # 抽取所选取值对应的数据，删去本属性对应列构成新数据集  # 重复以上操作，直到构成可接受的决策树为止 |

计算信息熵的函数如下：

|  |
| --- |
| def calcEntropy(所选属性,数据数量):  # 计算该属性每个取值的概率  # 计算每个取值的信息熵  # 取和得到该属性的信息熵 |

计算信息增益的函数如下：

|  |
| --- |
| def getInformationGain(信息熵,数据集):  # 遍历数据集的每列：  # 提取指定属性列  # 计算每个取值的概率  # 计算在每个取值的情况下，目标属性为yes的概率  # 计算每个取值下的信息熵  # 计算条件熵  # 计算信息增益 |

### 2.4 程序说明

dataInput.py为数据预处理，该程序读取元数据的csv文件，根据列名对数据进行处理，并保存枚举型数据处理后的映射关系。映射如下图所示：

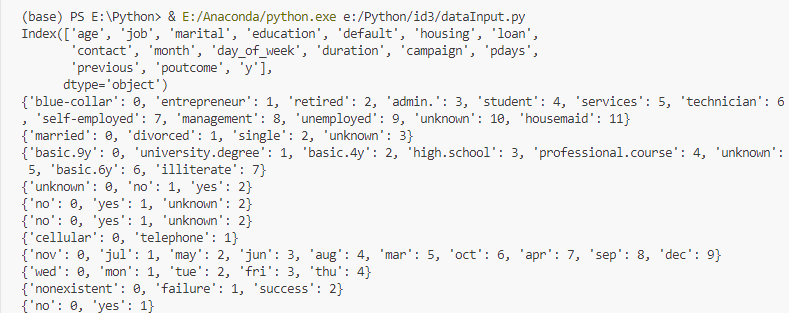


图 6 枚举类型数据处理后的映射

id3.py为算法主体，读取dataInput.py处理后的csv文件并执行id3算法。第一层决策树的生成是默认的，控制台会输出相应信息以帮助确定下一层决策树的根节点。如下图所示：



图 7 id3.py输出第一轮算法完成后所得信息

graph.py用于绘图。其引入pyecharts库，以json对象为元数据绘图。id3.py在执行时为每一层决策树输出一个符合树状图格式的result.json，graph.py读取并生成可交互的html页面，如下图所示：

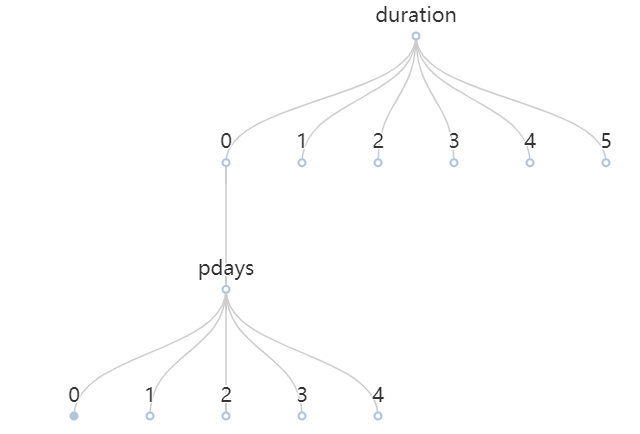


图 8 graph.py生成的树形图

recoverResultByMaps.py用于恢复枚举类型数据的原名称。dataInput.py在执行完成后会将枚举类型数据映射以json文件形式存储到本地，recoverResultByMaps.py处理id3.py存储的result.json文件，将映射的value重新转为key，如下图所示：

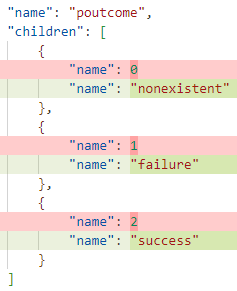


图 9 recoverResultByMaps.py的处理结果

### 2.5 时间和空间复杂度

设m为数据数量，n为属性数量。

id3算法计算信息增益需要对数据集的各列进行常数次遍历，具体次数与该列的可取值数量线性相关，因此该部分时间复杂度约为O(m \* n)。

id3算法构建决策树将对整个数据集进行递归处理，考虑每轮迭代属性数量-1，对每轮迭代的时间复杂度进行多项式求和，得时间复杂度为O(m \* n2)。考虑每轮迭代数据量减少，在最恶劣情况下每次都恰好将数据集二等分，则最终时间复杂度为O(m\*n2\*log(m))。

id3算法最大需要存储m行n列数据的空间，即空间复杂度为O(m \* n)。

id3算法生成完整的决策树最大有n层，每层的结点数取决于根结点的可取值数量。设各属性平均可取值数量为k，空间复杂度约为O(n\*kn-1)。

在实际的数据处理中，不需要生成完整的决策树，实际时间复杂度远低于O(m\*n2\*log(m))，实际空间复杂度中决策树部分占据的空间通常很小，可以忽略不计。

### 2.6 算法边界

id3算法仅适用于离散型数据，处理连续型数据需要先进行离散化处理。

id3算法在信息增益的选择上倾向于具有过多可取值数量的属性，因此适合处理属性可取值数量小且各取值区分度高的数据。

id3算法无法处理缺失值，适合处理完整的数据集，不适合处理模糊的数据集。

id3算法时间复杂度较高，处理大规模数据集时性能消耗显著增加，且随决策树深度增加过拟合现象严重，因此适合处理小规模，属性值较少的数据集。处理大规模数据集需要较多的人工处理。

在现实应用中，id3算法常用于信用评分与贷款决策、电子商务推荐、学生分类与成绩预测、人力资源管理等领域。

## 3 运行结果

数据预处理阶段已得知客户总体付费率为11.27%。考虑到业务实际情况，付费率在30%以上的分支就认为是优秀分支，不再向深层迭代。

第一轮运行情况如下图：

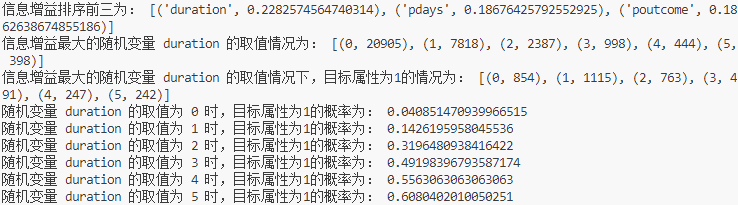


图 10 第一轮迭代完成情况

duration，即电话时长。由计算结果知，电话时长在12分钟及以上时，客户就有50-60%的几率订购产品。电话时长在8到12分钟之间时，产品订购率下降到约32%。电话时长在4到8分钟之间，产品订购率仅有约14%。

此处选取duration = 0 的分支进行第二轮迭代，情况如下图：

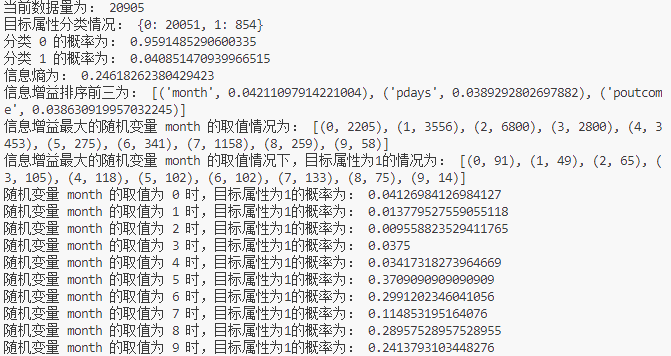


图 11 第二轮迭代完成情况

month，即与客户联系的月份。由计算结果知，在三月，九月，十月，十二月四个月份中客户订购产品的概率显著高于其他月份，达到25-37%。

有理由认为三月高峰期是由于该银行不在一、二月开展业务，需求积压到三月集中完成导致的。

但值得注意的是，这四个月份所对应的客户数量也是十个月份中明显偏少的四个月，与其他月份的人数有较大差距。为避免第一轮迭代筛去的数据影响结论，对原数据集的month属性进行分析，得到结果如下：

|  |  |  |  |
| --- | --- | --- | --- |
| 月份 | 联系人数 | 订购人数 | 订购率 |
| 三月 | 436 | 217 | 0.498 |
| 四月 | 2085 | 427 | 0.205 |
| 五月 | 11011 | 699 | 0.063 |
| 六月 | 4247 | 441 | 0.104 |
| 七月 | 5763 | 532 | 0.092 |
| 八月 | 4948 | 531 | 0.107 |
| 九月 | 464 | 207 | 0.446 |
| 十月 | 587 | 254 | 0.433 |
| 十一月 | 3266 | 332 | 0.102 |
| 十二月 | 143 | 72 | 0.503 |

表 1 原数据month属性分析结果

由分析结果知该现象具有普遍性，恢复映射后结合数据分析知，该银行在三月到五月间联系人数逐渐上升，在五月达到极高峰，在六到八月每月联系人数约为5月的50%，此后每月联系人数依次下降，只在十一月突增。经查询，葡萄牙法律规定薪水以14薪形式发放，每年圣诞节前发放完毕，且葡萄牙多在月底发放月薪，有理由认为正是该因素导致了十一月突然增加的联系人数。

基于以上分析结果，将mouth属性进行重新处理，划分为淡季月（订购率低于40%）和旺季月（订购率高于40%），新的分析结果如下：

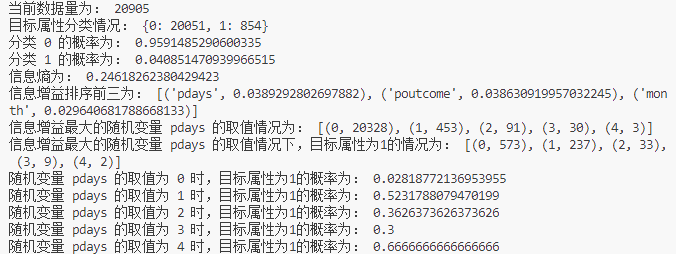


图 12 修改month属性后第二轮迭代完成情况

pdays，即自上次营销活动最后一次联系客户以来经过的天数。由计算结果知，在电话时长小于4分钟的情况下，从未联系过的客户第一次就订购产品的概率仅有不到3%，而之前联系过的客户在间隔小于一周的情况下，订购产品的概率高达52%，间隔时间介于一周至两周的情况下，订购率下降到36%（仍然较高）。间隔时间更长的数据样本过少，不具有代表性。

此处选取pdays = 0 的分支进行第三轮迭代，情况如下图：

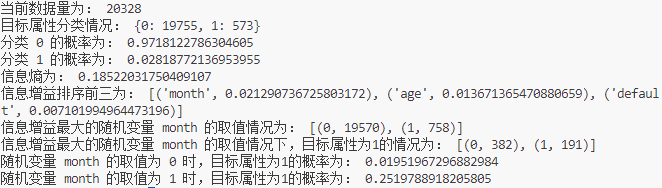


图 13 第三轮迭代完成情况

此时又选取了month属性，选取month = 0 的分支进行第四轮迭代，情况如下图：

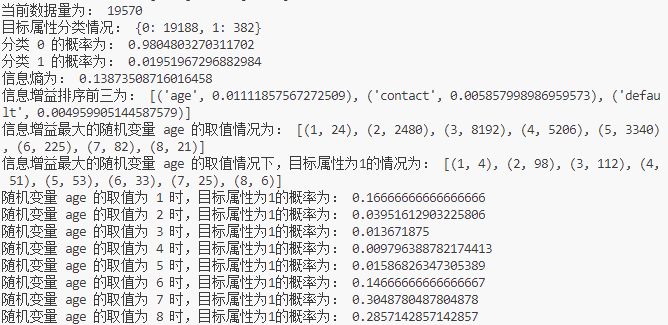


图 14 第四轮迭代完成情况

age，即年龄。由计算结果知，年龄在70岁以上的客户订购率达到29-30%，年龄在10-20岁之间的客户订购率达到17%（数据样本过少不具有代表性），年龄在60-70岁的客户订购率达到14%。其余年龄段客户订购率都低于4%，尤其40-50岁客户订购率不足1%。

选取age = 2、3、4、5、6分别进行第五轮迭代，情况如下图：

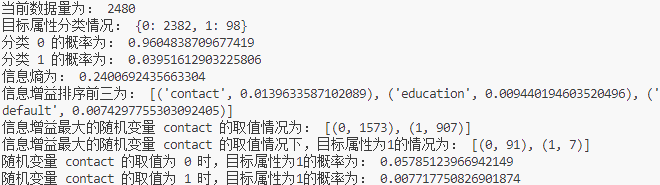


图 15 第五轮迭代完成情况-1

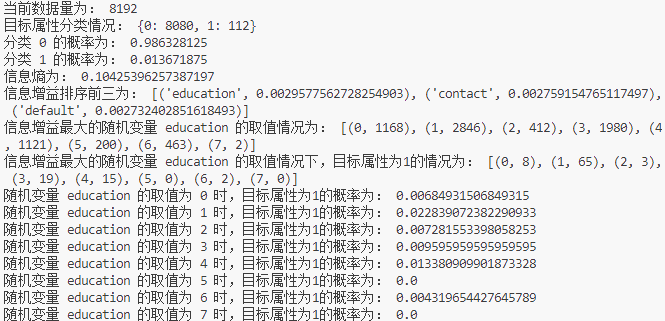


图 16 第五轮迭代完成情况-2

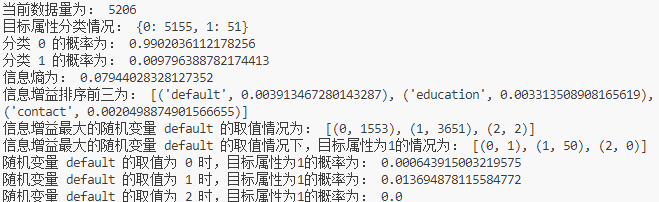


图 17 第五轮迭代完成情况-3

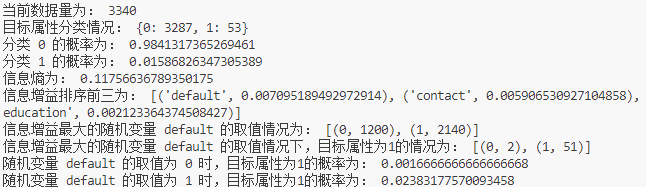


图 18 第五轮迭代完成情况-4

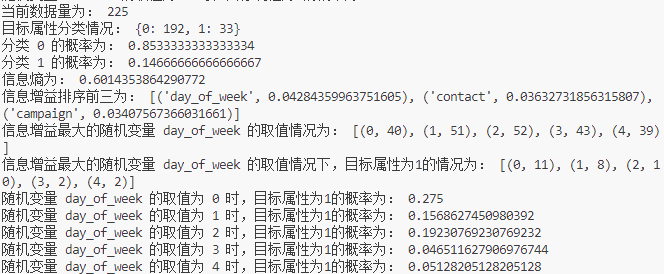


图 19 第五轮迭代完成情况-5

由计算结果知，20-30岁客户中影响最大的因素为contact，即联系方式。在两种通讯类型中，使用蜂窝通讯（实质上等同于使用苹果设备）的客户订购率达到5.8%，远高于使用移动电话的客户（0.8%）。

30-40岁客户中影响最大的因素为education，即受教育水平。在八种受教育水平中，最高的订购率是拥有大学学位的客户，订购率达到2.3%。其次是拥有职高教育水平的客户，订购率达到1.3%，其余教育水平的客户订购率均低于1%。值得注意的是，当客户不愿意表明自己的教育水平时（即取值为unknown），200名客户无一订购产品。

40-50岁客户中影响最大的因素为default，即信用违约状况。显而易见的，两名表明自己存在信用违约状况的客户未订购产品，而表明自己没有信用违约状况的客户订购率达1.3%，远高于不愿意表明自己信用违约状况的客户（1553名客户中仅有1人订购）。50-60岁客户中情况大致与40-50岁客户一致，且表明自己没有信用违约状况的客户订购率提高到2.4%。

60-70岁客户中影响最大的因素为day\_of\_week，即联系的日期是星期几。在周三联系客户，订购率达27.5%；在周二联系客户，订购率达19%；在周一联系客户，订购率达16%；而在周四周五联系客户，订购率迅速下降到5%左右。

至此，几乎所有分支的订购率都低于1%或数据样本过少，回溯至第三轮迭代，选取month = 1 的分支进行第四轮迭代，情况如下图：

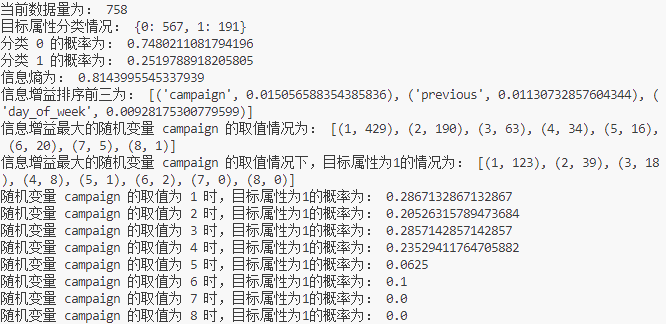


图 20 第四轮迭代完成情况-1

campaign，即联系次数。由计算结果知，联系客户的次数在四次及以内时，都有20%以上的订购率。但当联系次数超过四次后，订购率急剧下降（也与数据样本过少有关）。

回溯至第一轮迭代，选取duration = 1 的分支进行第二轮迭代，情况如下图：

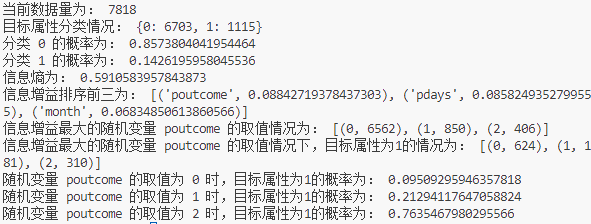


图 21 第二轮迭代完成情况-1

poutcome，即上次营销活动的结果。由计算结果知，在没有上次营销活动的情况下，用户订购产品的概率约9%，而上次举办过营销活动的情况下，即使活动结果为失败，用户订购产品的概率也上升到21%，而活动成功的情况下，用户订购产品的概率高达76%。

取 poutcome = 0、1 的分支进行第三轮迭代，情况如下图：

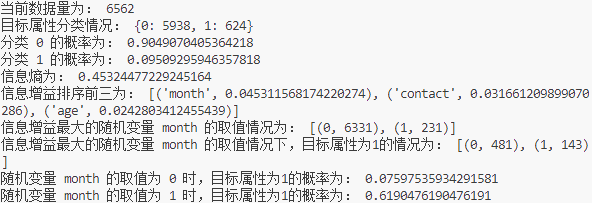


图 22 第三轮迭代完成情况-1

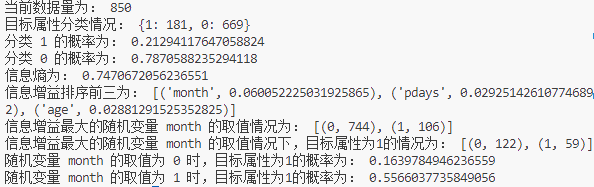


图 23 第三轮迭代完成情况-2

由计算结果知，淡季旺季对付费率影响较大，取mouth = 0的分支分别进行第四轮迭代，情况如下图：

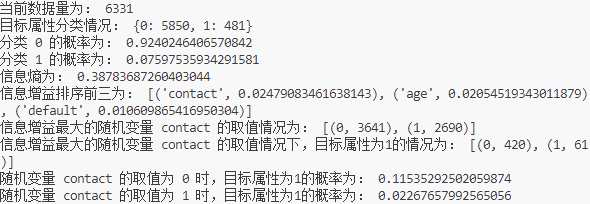


图 24 第四轮迭代完成情况-1

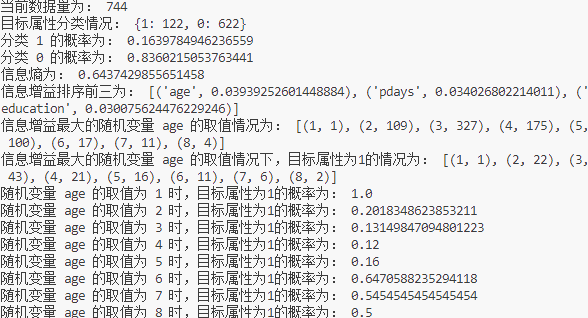


图 25 第四轮迭代完成情况-2

由计算结果知，在未接触过营销活动的客户中，contact因素影响较大。使用蜂窝通讯的客户付费率达到11.5%，远高于使用移动电话的客户（2%）。而在经历过失败的营销活动的客户中，age因素影响较大，60岁以上客户付费率在50-65%，20-60岁客户付费率只有12-20%。

对未接触过营销活动的客户分支，分别取contact = 0、1 的分支进行第五次迭代，情况如下图：

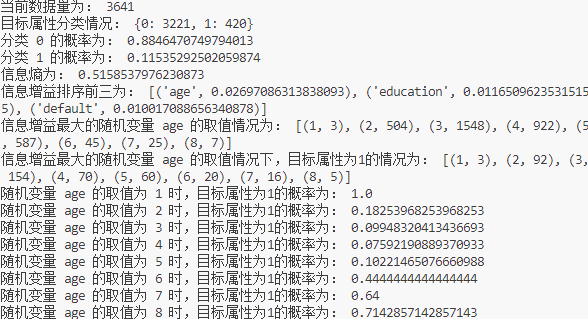


图 26 第五轮迭代完成情况-6

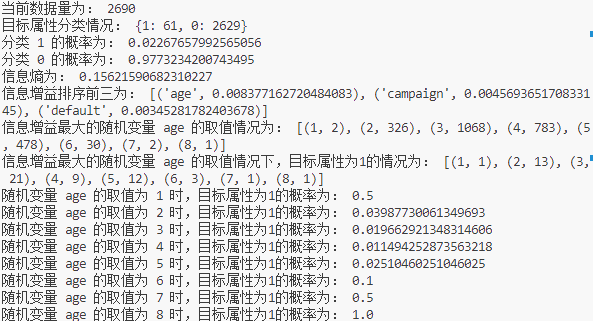
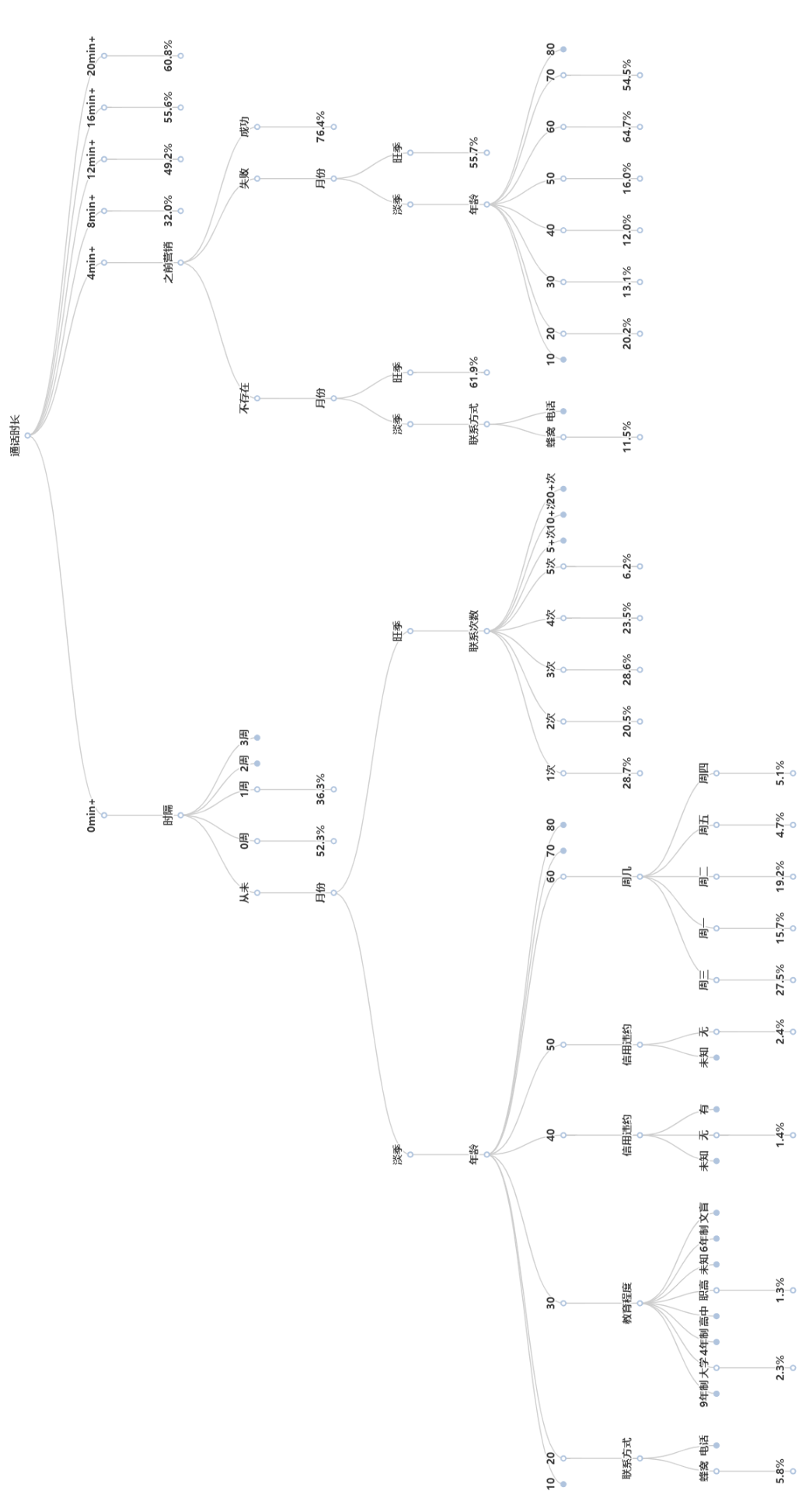


图 27 第五轮迭代完成情况-7

由计算结果知，在使用蜂窝通讯的客户中，20-30岁年轻群体的付费率高达18%，30-60岁客户付费率为7.5-10%，而使用移动电话的客户各年龄层付费率均低于5%，主要客户群体只有1-4%。

最终生成决策树如图（为便于展示部分结点未展开）：



## 4 结论

通过id3算法对数据集进行分析，可得到以下结论：

一、**客户付费率与电话时长高度相关。**与客户的通话时间在8分钟以上就可以认为该客户有较高可能性订购产品，因此对之前通话时间较长且尚未订购产品的客户要重点关注，在与客户交流时可考虑更加循序渐进的交流方式。可考虑对银行员工进行交流技巧培训，学习如何营造休闲放松的交流氛围，电话中避免刻意推荐储蓄产品，以深入沟通为主要交流方向。

二、**客户付费率与月份高度相关。**该银行在一年的十个工作月期间每个月的联系人数差异极大，却没有给出相应原因，相关因素难以分析。应考虑请该银行提供年度工作计划表，营销活动计划表等相关信息作为补充。

三、**短期（一周）内多次联系客户能有效提高付费率。**即使每次电话时长较短，在一周内联系客户两次以上仍然能显著提高付费率（甚至能取得接近20倍的效果），间隔时间超过一周效果就会下降约25%。

四、**高年龄客户付费率相对较高，中年龄群体付费率相对较低。**年龄在60岁以上的客户付费率就已经超越平均值，70岁以上的客户付费率更是达到30%左右，在拓展新客户时可以优先考虑高年龄群体。年龄在30-60岁的客户付费率显著低于平均值，在拓展新客户时应将该年龄群体列为低优先级。

五、**各年龄段的付费人群具有不同的属性特征。**20-30岁客户中使用蜂窝通讯（使用苹果设备）的客户订购率高，应给负责相应客户群体的员工部门配备蜂窝通讯设备；30-40岁客户中受教育水平为拥有大学学位和拥有职高教育水平的客户订购率高，可以考虑在大学与职业高中校园中举行活动以拓展客户，在线上调研客户画像时，应优先选取愿意表明自己教育水平的客户，不愿意表明自己的教育水平的客户可基本不予考虑；40-60岁客户中表明自己没有信用违约状况的客户订购率高，而隐瞒自己的信用状况的客户与表明自己存在信用违约状况的客户应当直接放弃，不予考虑；60-70岁客户在周一到周三时联系订购率较高，尽量避免在周四周五进行联系。

六、**客户联系次数不宜超过四次。**根据分析结果，联系客户的次数在四次以内能保证20%以上的订购率，但联系次数超过四次后订购率急剧下降，联系次数超过四次的客户除特殊情况外可考虑放弃。

七、**举办营销活动无论是失败还是成功都能对付费率起到促进作用。**在举办了营销活动的情况下，即使活动失败，参与过活动的客户付费率也上涨了两倍左右，而成功的活动对付费率的促进作用极大（上涨八倍）。因此银行举办营销活动不能只以成功失败为导向，要建立规律的营销活动计划，在此基础上尽可能提高活动成功率。

八、**优先考虑接触过成功的营销活动的用户；其次优先考虑接触过失败的营销活动的客户，其中又以高年龄客户优先；最后考虑未接触过营销活动的客户，首先考虑使用蜂窝通讯的客户，其中又以20-30岁年轻客户为最优先。**在营销活动时期考虑到任务较为紧张，可按照该顺序先联系付费可能性较高的用户。**5 附录**

附录A 核心代码

id3.py

1. import pandas as pd
2. import numpy as np
3. import os
4. import matplotlib.pyplot as plt
5. import networkx as nx
6. import json
7. path = os.path.join(os.path.dirname(\_\_file\_\_), 'train\_ready.csv')
8. data = pd.read\_csv(path)
9. data = data.drop(['job'], axis=1)
10. total = len(data)
11. targetIndex = data['y']
12. IndexValues = {}
13. IndexHs = {}
14. def calcEntropy(targetIndex,total):
15. p = {}
16. for c in targetIndex:
17. if c not in p:
18. p[c] = 1
19. else:
20. p[c] += 1
22. entropy = 0
23. for c in p:
24. entropy += -p[c] / total \* np.log2(p[c] / total)
25. return entropy
26. def calcConditionalEntropy(data,index):
27. values = data[index]
28. thisIndexValue = {}
29. for v in values:
30. if v not in thisIndexValue:
31. thisIndexValue[v] = 1
32. else:
33. thisIndexValue[v] += 1
35. IndexValues[index] = thisIndexValue
37. thisIndexH = {}
38. tempData = data[[index, 'y']]
39. for v in tempData.values:
40. if v[1] == 1:
41. if v[0] not in thisIndexH:
42. thisIndexH[v[0]] = 1
43. else:
44. thisIndexH[v[0]] += 1
46. for v in thisIndexValue:
47. if v not in thisIndexH:
48. thisIndexH[v] = 0
49. IndexHs[index] = thisIndexH
51. everyValueEntropy = {}
52. for v in thisIndexValue:
53. p = thisIndexH[v] / thisIndexValue[v]
54. if p == 0 or p == 1:
55. everyValueEntropy[v] = 0
56. else:
57. everyValueEntropy[v] = -p \* np.log2(p) - (1 - p) \* np.log2(1 - p)
59. conditionalEntropy = 0
60. for v in everyValueEntropy:
61. conditionalEntropy += thisIndexValue[v] / len(data) \* everyValueEntropy[v]
62. return conditionalEntropy
63. def calcInformationGain(entropy, conditionalEntropy):
64. return entropy - conditionalEntropy
65. def getInformationGain(entropy,data):
66. IGs = {}
67. for index in data.columns[:-1]:
68. conditionalEntropy = calcConditionalEntropy(data,index)
69. informationGain = calcInformationGain(entropy, conditionalEntropy)
70. IGs[index] = informationGain
71. return IGs
73. def outputGraphData(IGs,num):
74. IGs = pd.DataFrame(IGs)
75. IGs.to\_csv(os.path.join(os.path.dirname(\_\_file\_\_), 'Gain' + str(num) + '.csv'), index=False)
76. def saveGraphData(IGs,num,IndexValue,IndexH):
77. p = {}
78. for item in zip(IndexValue, IndexH):
79. p[item[0][0]] = "{:.1%}".format(item[1][1] / item[0][1])
80. graphData = []
81. temp = []
82. for item in IndexValues[IGs[0][0]]:
83. t = {
84. "name": item[0],
85. "children": [
86. {
87. "name": p[item[0]],
88. }
89. ]
90. }
91. temp.append(t)
92. graphData.append({"name": IGs[0][0], "children": temp})
94. with open(os.path.join(os.path.dirname(\_\_file\_\_), 'result' + str(num) + '.json'), 'w') as f:
95. json.dump(graphData, f)
96. def sortIndexValuesAndIndexHs(IGs):
97. IndexValues[IGs[0][0]] = sorted(IndexValues[IGs[0][0]].items(), key=lambda x: x[0])
98. IndexHs[IGs[0][0]] = sorted(IndexHs[IGs[0][0]].items(), key=lambda x: x[0])
99. def id3(targetColumn,data,num):
100. entropy = calcEntropy(data['y'], len(data))
101. IGs = getInformationGain(entropy, data)
102. IGs = sorted(IGs.items(), key=lambda x: x[1], reverse=True)
103. outputGraphData(IGs,num)
104. sortIndexValuesAndIndexHs(IGs)
105. saveGraphData(IGs,num,IndexValues[IGs[0][0]],IndexHs[IGs[0][0]])
107. def main():
108. Indexs = data.columns[:-1]
109. id3(targetIndex,data,0)
110. *# 第二轮*
111. *# 提取duration为0的数据构成新的数据集*
112. newData1 = data[data['duration'] == 0]
113. newData1 = newData1.drop(['duration'], axis=1)
115. id3(newData1['y'],newData1,1)
117. *# 第三轮*
118. *# 提取pdays为0的数据构成新的数据集*
119. newData2 = newData1[data['pdays'] == 0]
120. newData2 = newData2.drop(['pdays'], axis=1)
122. id3(newData2['y'],newData2,2)
123. *# 第四轮*
124. *# 提取month为0的数据构成新的数据集*
125. newData3 = newData2[data['month'] == 0]
126. newData3 = newData3.drop(['month'], axis=1)
128. id3(newData3['y'],newData3,3)
130. *# 第五轮*
131. *# 提取age = 1、2、3、4、5、6的数据，分别构成新的数据集*
132. newData4\_1 = newData3[newData3['age'] == 2]
133. newData4\_1 = newData4\_1.drop(['age'], axis=1)
134. newData4\_2 = newData3[newData3['age'] == 3]
135. newData4\_2 = newData4\_2.drop(['age'], axis=1)
136. newData4\_3 = newData3[newData3['age'] == 4]
137. newData4\_3 = newData4\_3.drop(['age'], axis=1)
138. newData4\_4 = newData3[newData3['age'] == 5]
139. newData4\_4 = newData4\_4.drop(['age'], axis=1)
140. newData4\_5 = newData3[newData3['age'] == 6]
141. newData4\_5 = newData4\_5.drop(['age'], axis=1)
142. id3(newData4\_1['y'],newData4\_1,4.1)
143. id3(newData4\_2['y'],newData4\_2,4.2)
144. id3(newData4\_3['y'],newData4\_3,4.3)
145. id3(newData4\_4['y'],newData4\_4,4.4)
146. id3(newData4\_5['y'],newData4\_5,4.5)
148. *#####################################*
149. *#tip 第一次回溯到第三轮*
150. *# # 第四轮*
151. *# # 提取month为1的数据构成新的数据集*
152. *# newData3 = newData2[data['month'] == 1]*
153. *# newData3 = newData3.drop(['month'], axis=1)*
155. *# id3(newData3['y'],newData3,3)*
157. *######################################*
158. *#tip 第二次回溯到第一轮*
159. *# # 第二轮*
160. *# # 提取duration为1的数据构成新的数据集*
161. *# newData1 = data[data['duration'] == 1]*
162. *# newData1 = newData1.drop(['duration'], axis=1)*
164. *# id3(newData1['y'],newData1,1)*
166. *# # 第三轮*
167. *# # 提取poutcome为0、1的数据构成新的数据集*
168. *# newData2\_1 = newData1[newData1['poutcome'] == 0]*
169. *# newData2\_1 = newData2\_1.drop(['poutcome'], axis=1)*
170. *# newData2\_2 = newData1[newData1['poutcome'] == 1]*
171. *# newData2\_2 = newData2\_2.drop(['poutcome'], axis=1)*
173. *# id3(newData2\_1['y'],newData2\_1,2.1)*
174. *# id3(newData2\_2['y'],newData2\_2,2.2)*
176. *# # 第四轮*
177. *# # 分别提取两个数据集中的mouth为0的数据构成新的数据集*
178. *# newData3\_1 = newData2\_1[newData2\_1['month'] == 0]*
179. *# newData3\_1 = newData3\_1.drop(['month'], axis=1)*
180. *# newData3\_2 = newData2\_2[newData2\_2['month'] == 0]*
181. *# newData3\_2 = newData3\_2.drop(['month'], axis=1)*
183. *# id3(newData3\_1['y'],newData3\_1,3.1)*
184. *# id3(newData3\_2['y'],newData3\_2,3.2)*
186. *# # 第五轮*
187. *# # 提取newData3\_1中的contact为0、1的数据构成新的数据集*
188. *# newData4\_1 = newData3\_1[newData3\_1['contact'] == 0]*
189. *# newData4\_1 = newData4\_1.drop(['contact'], axis=1)*
190. *# newData4\_2 = newData3\_1[newData3\_1['contact'] == 1]*
191. *# newData4\_2 = newData4\_2.drop(['contact'], axis=1)*
193. *# id3(newData4\_1['y'],newData4\_1,4.1)*
194. *# id3(newData4\_2['y'],newData4\_2,4.2)*
195. if \_\_name\_\_ == '\_\_main\_\_':
196. main()