

## 1.实验内容：

对美国国会投票案例进行关联规则挖掘

1.数据来源：<http://archive.ics.uci.edu/ml/datasets/Congressional+Voting+Records>

2.使用 Apriori 算法，支持度设为 30%，置信度为 90%，挖掘高置信度的规则。

## 2.分析及设计：

1.数据集中投票的来源有两类人，一类是共和党人，另一类是民主党人。数据集中的投票一共有三种表现形式：赞成（y），反对（n）和弃权（?）。由于弃权占了实验中的数据集中的少数，因此在本次实验中我并没有去对弃权的票进行关联规则分析。因此选定了以下四种关联规则挖掘方式：

(1).对身份是共和党+投赞成票进行关联规则分析，选出哪些事件之间可能存在关联。

(2).对身份是共和党+投反对票进行关联规则分析，选出哪些事件之间可能存在关联。

(3).对身份是民主党+投赞成票进行关联规则分析，选出哪些事件之间可能存在关联。

(4).对身份是民主党+投反对票进行关联规则分析，选出哪些事件之间可能存在关联。

2.首先了解 Apriori 算法的原理：

Apriori 是一种常用的数据关联规则挖掘方法，它可以用来找出数据集中频繁出现的数据集合。找出这样的一些频繁集合有利于决策，例如通过找出超市购物车数据的频繁项集，可以更好地设计货架的摆放。需要注意的是它是一种**逐层迭代**的方法，先找出频繁 1 项集 L1，再利用 L1 找出频繁 2 项集，以此类推……

3.在使用 Apriori 算法进行设计时，需要考虑剪枝，及时排除掉支持度不足的规则，这样可以很有效地提升程序的性能。

## 3.详细实现：

对数据进行归类统计代码实现如下：（代码中附带注释）

```

1 import pandas as pd
2
3 mydata=pd.read_csv('/home/anqian/Test2/house-votes-84.data', sep=',', header=None)
4 #使用mydata进行投票结果的统计
5 republican_ycount=[]
6 republican_ncount=[]
7 republican_wcount=[]
8 democrat_ncount=[]
9 democrat_ycount=[]
10 democrat_wcount=[]
11 for i in range(16):
12     republican_ycount.append(0)
13     republican_ncount.append(0)
14     republican_wcount.append(0)
15     democrat_ycount.append(0)
16     democrat_ncount.append(0)
17     democrat_wcount.append(0)
18     for j in range(435):
19         if mydata[0][j] == 'republican':
20             if mydata[i+1][j]=='y':
21                 republican_ycount[i]+=1
22             elif mydata[i+1][j]=='?':
23                 republican_wcount[i]+=1
24             else:
25                 republican_ncount[i]+=1
26         else:
27             if mydata[i+1][j]=='y':
28                 democrat_ycount[i]+=1
29             elif mydata[i+1][j] == '?':
30                 democrat_wcount[i]+=1
31             else:
32                 democrat_ncount[i]+=1
33
34 #输出投票统计的结果
35 print('republican_y:',republican_ycount)
36 print('republican_n:',republican_ncount)
37 print('republican_w:',republican_wcount)
38 print('democrat_y:',democrat_ycount)
39 print('democrat_n:',democrat_ncount)
40 print('democrat_w:',democrat_wcount)

```

## Apriori 算法的实现：

```

def mining(data, dic_before, suport, coincidence):
    dic_3 = {}
    #对二阶遍历输出的字典进行挖掘
    for item0 in dic_before:
        for item1 in dic_before:
            if (item0 != item1):
                # print(item0,item1)
                item_len = len(item0)
                equal = True
                tmp_item3 = []
                # 判断前n-1项是否一致
                for i in range(item_len - 1):
                    tmp_item3.append(item0[i])
                    if (item0[i] != item1[i]):
                        equal = False
                        break
                if (equal == True):
                    minitem = min(item0[-1], item1[-1])
                    maxitem = max(item0[-1], item1[-1])
                    tmp_item3.append(minitem)
                    tmp_item3.append(maxitem)
                    tmp_item3 = tuple(tmp_item3)
                    dic_3[tmp_item3] = 0
                else:
                    continue
    # print('dic_3:',dic_3)
    #剪枝
    for data_line in data:
        for item in dic_3:
            is_in = True
            for i in range(len(item)):
                if (item[i] not in data_line):
                    is_in = False
                    break
            if (is_in == True):
                dic_3[item] += 1

    #计算数据规模, 用于计算支持度和置信度
    count = len(data)
    dic_3n = {}

    for item in dic_3:
        count_item0 = dic_before[item[:-1]]
        # 判断支持度和置信度
        if ((dic_3[item] >= suport * count) and (dic_3[item] >= coincidence * count_item0)):
            dic_3n[item] = dic_3[item]

    return dic_3n

```

需要注意的是：Apriori 算法更深层次关联规则的挖掘基于对关联规则进行初步的挖掘，代码如下：

```
def mining_first(data, suport, coincidence):
    #挖掘候选1-项集
    dic = {}
    count = len(data)
    for data_line in data:
        for data_item in data_line:
            if(data_item in dic):
                dic[data_item] += 1
            else:
                dic[data_item] = 1

    item_suport = int(count * suport)
    dic_1 = {}
    for item in dic:
        if(dic[item] >= item_suport):
            dic_1[item] = dic[item]

    return dic_1

def mining_second(data, dic_before, suport, coincidence):
    #挖掘出候选2-项集
    dic = {}
    count = len(data)
    count2 = 0
    for data_line in data:
        for i in range(len(data_line)):
            for j in range(i + 1, len(data_line)):
                if(data_line[i] in dic_before and data_line[j] in dic_before):
                    count2 += 1
                    tmp = (data_line[i], data_line[j])
                    if(tmp in dic):
                        dic[tmp] += 1
                    else:
                        dic[tmp] = 1
            else:
                continue

    dic_2 = {}
    for item in dic:
        count_item0 = dic_before[item[0]]
        count_item1 = dic_before[item[1]]
        # 判断支持度和置信度
        if((dic[item] >= suport * count) and ((dic[item] >= coincidence * count_item0 or (dic[item] >= coincidence * count_item1)))):
            dic_2[item] = dic[item]

    return dic_2
```

## 4.实验结果：

输出结果如图所示：

```
anqlan@anqlan-Lenovo-Legion-Y7000-2020: ~/桌面/DataMining/实验三$ python Code.py
republican_y: [31, 75, 22, 163, 157, 149, 39, 24, 19, 92, 21, 135, 136, 158, 14, 96]
republican_n: [134, 73, 142, 2, 8, 17, 123, 133, 146, 73, 138, 20, 22, 3, 142, 50]
republican_w: [3, 28, 4, 3, 3, 2, 6, 11, 3, 3, 9, 13, 10, 7, 12, 22]
democrat_y: [156, 120, 231, 14, 55, 123, 200, 218, 188, 124, 129, 36, 73, 90, 160, 173]
democrat_n: [102, 119, 29, 245, 200, 135, 59, 45, 60, 139, 126, 213, 179, 167, 91, 12]
democrat_w: [9, 28, 7, 8, 12, 9, 8, 4, 19, 4, 12, 18, 15, 10, 16, 82]
republican_vote YES:
[[{(2, 4): 75, (2, 5): 73, (2, 6): 72, (2, 13): 68, (4, 5): 156, (4, 6): 147, (4, 10): 91, (4, 12): 134, (4, 13): 135, (4, 14): 155, (4, 16): 94, (5, 6): 145, (5, 10): 85, (5, 12): 131, (5, 13): 132, (5, 14): 14, (5, 16): 89, (6, 12): 127, (6, 13): 127, (6, 14): 140, (10, 14): 88, (12, 14): 129, (13, 14): 130, (14, 16): 94}, {(2, 4, 5): 73, (2, 4, 6): 72, (2, 4, 13): 68, (2, 5, 6): 71, (2, 5, 13): 67, (2, 6, 13): 66, (4, 5, 6): 144, (4, 5, 14): 148, (4, 6, 14): 139, (4, 10, 14): 87, (4, 12, 14): 129, (4, 13, 14): 130, (5, 6, 14): 137, (5, 10, 14): 81, (5, 12, 14): 126, (5, 13, 14): 127, (6, 12, 14): 121, (6, 13, 14): 121}, {(2, 4, 5, 6): 71, (2, 4, 5, 13): 67, (2, 4, 6, 13): 66, (2, 5, 6, 13): 65, (4, 5, 6, 14): 136}, {(2, 4, 5, 6, 13): 65}]
republican_vote No:
[[{(3, 7): 116, (3, 8): 123, (3, 9): 132, (7, 8): 116, (7, 9): 119, (7, 15): 114, (8, 9): 128, (8, 15): 121, (3, 10): 67, (9, 10): 69}, {(3, 7, 8): 111, (3, 7, 9): 115, (3, 8, 9): 121, (7, 8, 9): 114, (7, 8, 15): 108, (7, 9, 15): 111, (8, 9, 15): 116}, {(3, 7, 8, 9): 110, (7, 8, 9, 15): 106}]
democrat_vote Yes:
[[{(1, 3): 142, (3, 15): 144, (3, 7): 184, (3, 8): 203, (3, 9): 171, (7, 8): 189, (8, 9): 179, (3, 10): 112, (8, 15): 145}, {(3, 7, 8): 176}]
democrat_vote No:
[[{(4, 12): 201, (4, 14): 163, (4, 11): 123, (4, 5): 194, (4, 6): 133, (4, 13): 176, (5, 6): 132, (5, 14): 152, (6, 12): 123, (2, 4): 117}, {(4, 6, 12): 122}]
anqlan@anqlan-Lenovo-Legion-Y7000-2020: ~/桌面/DataMining/实验三$
```

上半部分的输出结果表示对结果统计的输出。

下半部分输出结果表示挖掘出的关联规则，以最下面一行的 democrat\_vote No 为例，(4,12) 表示第四个事件与第 12 个事件具有关联，换言之，民主党对“通过预算的决议”（事件 4）投了反对票，往往也会对“合成燃料公司削减”（事件 12）投反对票。

输出的实验结果不是很符合预期，比如 republic\_vote Yes 部分，有很多具有包含关系的频繁项集，在输出之前其实可以进行一些合并操作，但是由于最近各学科都很忙碌，进行数据挖掘的实验时间不是很充分，因此没有足够的时间去研究代码了，所以留有一些漏洞。

## 5.心得体会：

本次实验比较相比于上两次实验还是要复杂一些的，因为前两次实验可以自己选择调用 Python 的包，而本次实验只能选择自己实现，因此花的时间要比较多，通过对 Apriori 算法的实现，自己也是对该算法更加了解了，虽然没有实现的比较好，但是也算是一个初步的进展吧。

数据挖掘的实验就此完结了，在此谈一下总的感受吧。总体来说实验是真的帮助自己学到了很多实用的知识，比如实验一可以用类似的模型来训练自己想要分类的事物，把实验模型运用到生活中来，只要有充足的数据支持，训练的模型会更加的精确。自己对数据挖掘/机器学习的认识也有了一个比较完整的轮廓，如果想要提升自己的能力，需要做的可能就是更多的实践了。