实验介绍

1.实验内容

本实验介绍Apriori算法。并使用该算法来发现投票规则。

2.实验目标

通过本实验掌握Apriori算法。

3.实验知识点

- Apriori算法

4.实验环境

- python 3.6.5

5.预备知识

- 初等数学知识
- Linux命令基本操作
- Python编程基础

准备工作

点击屏幕右上方的下载实验数据模块，选择下载apriori_vote.tgz到指定目录下，然后再依次选择点击上方的File->Open->Upload,上传刚才下载的数据集压缩包，再使用如下命令解压：

In [23]:
```
!tar -zxvf apriori_vote.tgz
```

```
tar: Error opening archive: Failed to open 'apriori_vote.tgz'
```

【实验】收集数据

本节我们将在真实数据上使用Apriori算法。下面我们来看一个很有趣的例子，关于美国国会议员投票。我们将根据政客的行为来寻找投票规则，进而可以预测一个政客的投票。或者可以发现更有趣的关联关系。

需要的数据集为数据集目录下的bills20DataSet.txt文件，该文件内容如下：

In [16]:
```
!cat apriori_vote / bills20DataSet.txt
```

```
'cat' 不是内部或外部命令，也不是可运行的程序
或批处理文件。
```

可以看到，我们收集了每个政客的投票行为，包括政客本身的党性，以及对各种议题的投票结果，我们以整型数据来标识每个行为。

在meaning20.txt文件中，展示了每个整型数据所代表的具体意义。每行的行号即数据值。

In [17]:
```
!cat apriori_vote / meaning20.txt
```

```
'cat' 不是内部或外部命令，也不是可运行的程序
或批处理文件。
```

如0是民主党，1是共和党。

【实验】数据处理

首先，我们需要对数据进行处理，将数据转换成集合列表形式。添加如下代码：

In [18]:
```
def loadDataSet():
    """
    函数说明：对整型数据进行处理
    parameters:
        null
    return:
        dataSet -列表数据集
    """
    fileline = open('data/bills20DataSet.txt').readlines()
    dataSet = []
    listDataSet = []
    inLine = 0
    for line in fileline:
        line = line.split()
        if '(' in line[0]:
            if inLine == 0:
                inLine = 1
            else:
                dataSet.append(listDataSet)
                listDataSet = []
            continue
        listDataSet.extend(line)
    return dataSet


def loadMeaningData():
    """
    函数说明：存储数据标识信息
    parameters:
        null
    return:
        meanData -列表数据集对应的意义
    """
    fileline = open('data/meaning20.txt').readlines()
    meanData = []
    for line in fileline:
        line = line.split()
        meanData.append(line)
    return meanData
```

我们可以看下，数据处理后的效果，添加main函数：

In [19]:
```
if __name__ == '__main__':
    dataSet = loadDataSet()
    meanData = loadMeaningData()
    print(dataSet)
```

```
[['0', '7', '9', '10', '12', '14', '21', '23', '25', '26', '28'], ['0', '3', '4', '7', '9', '10', '13', '14', '16', '21', '23', '25',
'26', '28'], ['0', '3', '4', '7', '9', '23', '25', '26'], ['0', '3', '4', '7', '9', '10', '13', '14', '16', '21', '23', '25', '26', '2
8'], ['1', '2', '5', '7', '8', '11', '13', '15', '17', '21', '22', '24', '27', '29'], ['1', '2', '5', '6', '8', '13', '17', '21', '22',
'24', '27', '29'], ['0', '3', '4', '7', '9', '23', '25', '26'], ['1', '2', '5', '6', '8', '11', '13', '15', '17', '21', '22', '24', '2
7', '29'], ['1', '2', '5', '7', '8', '11', '13', '15', '17', '21', '22', '24', '27', '29'], ['1', '2', '4', '6', '8', '11', '13', '15',
'17', '21', '22', '25', '26', '29'], ['0', '3', '4', '7', '9', '10', '13', '15', '17', '21', '23', '25', '26', '28'], ['1', '11', '15',
'17', '20'], ['0', '3', '4', '7', '9', '23', '25', '26'], ['0', '3', '4', '7', '9', '11', '12', '14', '16', '21', '23', '25', '26', '2
8'], ['1', '11', '13', '15', '17', '21', '29'], ['0', '3', '4', '9', '10', '12', '14', '16', '23', '25', '26', '28'], ['1', '2', '4',
'6', '8', '11', '13', '15', '17', '21', '22', '24', '27', '29'], ['0'], ['1', '11', '13', '15', '16', '21', '28'], ['0', '3', '4', '6',
'9', '13', '14', '17', '21', '23', '25', '26', '28'], ['0', '7', '9', '10', '12', '14', '16', '21', '23', '25', '26', '28'], ['2', '5',
'6', '8', '11', '13', '15', '17', '20', '22', '24', '27', '29'], ['0', '3', '4', '7', '9', '10', '13', '14', '16', '21', '23', '25', '2
7', '28'], ['1', '2', '5', '6', '8', '11', '13', '15', '17', '21', '22', '24', '27', '29'], ['0', '3', '4', '7', '9', '10', '12', '14',
'16', '21', '23', '25', '26', '28'], ['0', '3', '4', '7', '9', '10', '12', '14', '16', '21', '23', '25', '26', '28'], ['1', '11', '13',
'15', '17', '21', '28'], ['1', '2', '5', '6', '8', '11', '13', '15', '17', '21', '22', '24', '27', '29'], ['1', '2', '4', '6', '8', '1
1', '13', '15', '17', '21', '22', '24', '26', '29'], ['0'], ['1', '2', '4', '6', '8', '11', '13', '15', '17', '21', '22', '24', '27',
'29'], ['1', '2', '4', '6', '8', '11', '13', '14', '17', '21', '23', '25', '26', '29'], ['1', '2', '4', '6', '8', '11', '13', '15', '1
7', '21', '22', '24', '27', '29'], ['1', '2', '5', '6', '8', '13', '15', '17', '21', '22', '24', '27', '29'], ['1', '11', '13', '15',
'17', '21', '29'], ['1', '11', '13', '15', '17', '21', '29'], ['0', '3', '4', '7', '9', '23', '25', '27'], ['1', '2', '5', '6', '8', '1
1', '13', '15', '17', '21', '22', '24', '27', '29'], ['0', '3', '4', '7', '9', '10', '12', '14', '16', '21', '23', '25', '26', '28'],
```

这就是我们要使用的数据。

【练习】测试算法

接下来，我们将应用上节的Apriori算法来进行处理。添加如下代码：

In [22]:

```
#注：forzenset为不可变集合，它的值是不可变的，好处是它可以作为字典的key，也可以作为其他集合的元素。
def createC1(dataSet):
    """
    函数说明：构建集合C1。即所有候选项元素的集合。
    parameters:
        dataSet -数据集
    return:
        frozenset列表
    output:
        [forzenset([1]),forzenset([2]),……]
    """
    C1 = []  #创建一个空列表
    for transaction in dataSet:  #对于数据集中的每条记录
        for item in transaction:  #对于每条记录中的每一个项
            if not [item] in C1:  #如果该项不在C1中，则添加
                C1.append([item])
    C1.sort()  #对集合元素排序
    return list(map(frozenset, C1))  #将C1的每个单元列表元素映射到forzenset()


def scanD(D, Ck, minSupport):
    """
    函数说明：构建符合支持度的集合Lk
    parameters:
        D -数据集
        Ck -候选项集列表
        minSupport -感兴趣项集的最小支持度
    return:
        retList -符合支持度的频繁项集合列表L
        supportData -最频繁项集的支持度
    """
    ssCnt = {}  #创建空字典
    for tid in D:  #遍历数据集中的所有交易记录
        for can in Ck:  #遍历Ck中的所有候选集
            if can.issubset(tid):  #判断can是否是tid的子集
                if not can in ssCnt:
                    ssCnt[can] = 1  #如果是记录的一部分，增加字典中对应的计数值。
                else:
                    ssCnt[can] += 1
    numItems = float(len(D))  #得到数据集中交易记录的条数
    retList = []  #新建空列表
    supportData = {}  #新建空字典用来存储最频繁集和支持度
    for key in ssCnt:
        support = ssCnt[key] / numItems  #计算每个元素的支持度
```

```
        if support >= minSupport:  #如果大于最小支持度则添加到retList中
            retList.insert(0, key)
        supportData[key] = support  #并记录当前支持度，索引值即为元素值
    return retList, supportData


def aprioriGen(Lk, k):
    """
    函数说明:构建集合Ck
    parameters:
        Lk -频繁项集列表L
        k -候选集的列表中元素项的个数
    return:
        retList -候选集项列表Ck
    """
    retList = []  #创建一个空列表
    lenLk = len(Lk)  #得到当前频繁项集合列表中元素的个数
    for i in range(lenLk):  #遍历所有频繁项集合
        for j in range(i + 1, lenLk):  #比较Lk中的每两个元素，用两个for循环实现
            L1 = list(Lk[i])[:k - 2];
            L2 = list(Lk[j])[:k - 2]  #取该频繁项集合的前k-2个项进行比较
            #[注]此处比较了集合的前面k-2个元素，需要说明原因
            L1.sort();
            L2.sort()  #对列表进行排序
            if L1 == L2:
                retList.append(Lk[i] | Lk[j])  #使用集合的合并操作来完成 e.g.: [0,1],[0,2]->[0,1,2]
    return retList


def apriori(dataSet, minSupport=0.5):
    """
    函数说明: apriori算法实现
    parameters:
        dataSet -数据集
        minSupport -最小支持度
    return:
        L -候选项集的列表
        supportData -项集支持度
    """
    C1 = createC1(dataSet)
    D = list(map(set, dataSet))  #将数据集转化为集合列表
    L1, supportData = scanD(D, C1, minSupport)  #调用scanD()函数，过滤不符合支持度的候选项集
    L = [L1]  #将过滤后的L1放入L列表中
    k = 2  #最开始为单个项的候选集，需要多个元素组合
    while len(L[k - 2]) > 0:
        Ck = aprioriGen(L[k - 2], k)  #创建Ck
```

```
        Lk, supK = scanD(D, Ck, minSupport)  #由Ck得到Lk
        supportData.update(supK)  #更新支持度
        L.append(Lk)  #将Lk放入L列表中
        k += 1  #继续生成L3, L4...
    return L, supportData


def generateRules(L, supportData, minConf=0.7):
    """
    函数说明: 关联规则生成函数
    parameters:
        L -频繁项集合列表
        supportData -支持度字典
        minConf -最小可信度
    return:
        bigRuleList -包含可信度的规则列表
    """
    bigRuleList = []  #创建一个空列表
    for i in range(1, len(L)):  #遍历频繁项集合列表
        for freqSet in L[i]:  #遍历频繁项集合
            H1 = [frozenset([item]) for item in freqSet]  #为每个频繁项集合创建只包含单个元素集合的列表H1
            if i > 1:  #要从包含两个或者更多元素的项集开始规则构建过程
                rulesFromConseq(freqSet, H1, supportData, bigRuleList, minConf)
            else:  #如果项集中只有两个元素, 则计算可信度值, (len(L)=2)
                calcConf(freqSet, H1, supportData, bigRuleList, minConf)
    return bigRuleList


def rulesFromConseq(freqSet, H, supportData, brl, minConf=0.7):
    """
    函数说明: 规则构建函数
    parameters:
        freqSet -频繁项集合
        H -可以出现在规则右部的元素列表
        supportData -支持度字典
        brl -规则列表
        minConf -最小可信度
    return:
        null
    """
    m = len(H[0])  #得到H中的频繁集大小m
    if len(freqSet) > (m + 1):  #查看该频繁集是否大到可以移除大小为m的子集
        Hmp1 = aprioriGen(H, m + 1)  #构建候选集Hm+1, Hmp1中包含所有可能的规则
        Hmp1 = calcConf(freqSet, Hmp1, supportData, brl, minConf)  #测试可信度以确定规则是否满足要求
        if len(Hmp1) > 1:  #如果不止一条规则满足要求, 使用函数迭代判断是否能进一步组合这些规则
            rulesFromConseq(freqSet, Hmp1, supportData, brl, minConf)
```

```python
def calcConf(freqSet, H, supportData, brl, minConf=0.7):
    """
    函数说明：计算规则的可信度，找到满足最小可信度要求的规则
    parameters:
        freqSet -频繁项集合
        H -可以出现在规则右部的元素列表
        supportData -支持度字典
        brl -规则列表
        minConf -最小可信度
    return:
        prunedH -满足要求的规则列表
    """
    prunedH = []  #为保存满足要求的规则创建一个空列表
    for conseq in H:
        conf = supportData[freqSet] / supportData[freqSet - conseq]  #可信度计算[support(PUH)/support(P)]
        if conf >= minConf:
            print(freqSet - conseq, '-->', conseq, '可信度为:', conf)
            brl.append((freqSet - conseq, conseq, conf))  #对bigRuleList列表进行填充
            prunedH.append(conseq)  #将满足要求的规则添加到规则列表
    return prunedH


if __name__ == '__main__':
    dataSet = loadDataSet()
    meanData = loadMeaningData()
    L, suppData = apriori(dataSet, minSupport=0.3)
    rules = generateRules(L, suppData, minConf=1)
    while True:
        str = input("请输入想要知道的规则编号: ")
        if str == '':
            break
        print(meanData[int(str)])
```

```
frozenset({'3'}) --> frozenset({'9'}) 可信度为: 1.0
frozenset({'3', '26'}) --> frozenset({'9', '0'}) 可信度为: 1.0
frozenset({'3', '26', '4'}) --> frozenset({'9', '0'}) 可信度为: 1.0
frozenset({'3', '26', '7'}) --> frozenset({'9', '0'}) 可信度为: 1.0
frozenset({'3', '25', '26'}) --> frozenset({'9', '0'}) 可信度为: 1.0
frozenset({'3', '26', '23'}) --> frozenset({'9', '0'}) 可信度为: 1.0
frozenset({'3', '25', '23', '26'}) --> frozenset({'9', '0'}) 可信度为: 1.0
frozenset({'3', '25', '26', '7'}) --> frozenset({'9', '0'}) 可信度为: 1.0
frozenset({'3', '26', '23', '7'}) --> frozenset({'9', '0'}) 可信度为: 1.0
frozenset({'3', '26', '23', '4'}) --> frozenset({'9', '0'}) 可信度为: 1.0
frozenset({'3', '25', '26', '4'}) --> frozenset({'9', '0'}) 可信度为: 1.0
frozenset({'3', '26', '4', '7'}) --> frozenset({'9', '0'}) 可信度为: 1.0
frozenset({'4', '7', '26', '23', '3'}) --> frozenset({'9', '0'}) 可信度为: 1.0
frozenset({'4', '25', '26', '23', '3'}) --> frozenset({'9', '0'}) 可信度为: 1.0
frozenset({'4', '7', '25', '26', '3'}) --> frozenset({'9', '0'}) 可信度为: 1.0
frozenset({'7', '25', '26', '23', '3'}) --> frozenset({'9', '0'}) 可信度为: 1.0
['Democratic']
['Prohibiting', 'Federal', 'Funding', 'of', 'National', 'Public', 'Radio', '--', 'Nay']
['Prohibiting', 'Federal', 'Funding', 'of', 'National', 'Public', 'Radio', '--', 'Yea']
```

在main函数中，我们设置了一个循环，可以从生成的规则中知道对应序号的内容。运行后，我们可以看到生成的关联规则，并可以查询相应序号的意义。

数据中还有很多有趣的规则。还记得最开始使用的支持度30%吗？也就意味着这些规则至少出现在30%以上的记录中，这是很有意义的。对于规则{3,26}->{9,0}这条规则，在100%的情况下成立。这是很神奇的事情啊。

实验总结

本节我们介绍了Apriori算法，并介绍了如何使用Apriori算法，您应该能达到以下两个目标：

1. 掌握Apriori原理。
2. 学会使用该算法处理数据。

参考文献与延伸阅读

参考资料:

1.哈林顿，李锐. 机器学习实战 : Machine learning in action[M]. 人民邮电出版社, 2013.

2.周志华. 机器学习:Machine learning[M]. 清华大学出版社, 2016.

延伸阅读

1.李航. 统计学习方法[M]. 清华大学出版社, 2012.