**数据挖掘大作业二：关联规则挖掘**

学院：计算机学院 学号：2120151036 姓名：王丹

1. **任务说明**

本次统计任务是在标准数据集上进行关联规则的抽取与分析，从而发现有价值的规则。在对标准数据集预处理部分和抽取规则与评价的部分使用Java编写，采用了Apriori算法。可视化部分采用R编写。

1. **数据集介绍**
   1. 数据来源

使用了泰坦尼克的机器学习数据集，其中训练数据有近900条，测试数据有400余条。数据来源于网站：

<https://www.kaggle.com/c/titanic/data>

* 1. 数据样例：

[PassengerId,Survived,Pclass,Name,Sex,Age,SibSp,Parch,Cabin,Embarked,Ticket,Fare]

[1,0,3,"Braund, Mr. Owen Harris",male,22,1,0,,S,A/5 21171,7.25]

* 1. 字段说明：

表一 数据字段说明

|  |  |
| --- | --- |
| 字段 | 说明 |
| survived | Survival  (0 = No; 1 = Yes) |
| pclass | Passenger Class  (1 = 1st; 2 = 2nd; 3 = 3rd) |
| name | Name |
| Sex | Sex |
| Age | Age |
| sibsp | Number of Siblings/Spouses Aboard |
| parch | Number of Parents/Children Aboard |
| cabin | Cabin |
| embarked | Port of Embarkation  (C = Cherbourg; Q = Queenstown; S = Southampton) |
| ticket | Ticket Number |
| fare | Passenger Fare |

1. **预处理**
2. 删除信息含量低的字段

在关联规则挖掘中，我认为，乘客ID，票号，费用以及姓名信息、房产对规则没有任何帮助，故将其去除，只对其他属性进行关联规则分析。使用LiberOffice Calc删除。

1. 替换部分数值内容

在Survive字段中，将0替换为为Dead，1替换为Survive。

在Pclass字段中，分别将数字替换为坐席等级：1 = 1class; 2 = 2class; 3 = 3class。

对年龄，进行了划分。0-12岁为child，12-20为teenager，20-40为youth，40-60为middle-aged，60以上为elderly。

亲属情况，按照是否为0，简单的分为了有和无两种情况。

对于缺失项，因其对关联规则抽取意义不大，所以直接去除。

预处理后的数据样例：

[Survived,Pclass,Sex,Age,SibSp,Parch,Embarked]

BufferedReader reader = new BufferedReader(new FileReader("./train.csv"));

reader.readLine();//第一行信息，为标题信息，不用,如果需要，注释掉

File csv = new File("./predata.txt"); // txt数据文件

BufferedWriter bw = new BufferedWriter(new FileWriter(csv, true)); // 附加

String line = null;

double age = 0;

while ((line = reader.readLine()) != null) {

if (line.contains(",,")) continue;//去除不完全项

String item[] = line.split(",");//CSV格式文件为逗号分隔符文件，这里根据逗号切分

if(item.length!=7) continue;

if (item[0].equals("0")) { item[0] = "Dead"; }

else { item[0] = "Survived"; }

item[1] += "class";

age = Double.parseDouble(item[3]);

if (age <= 12) {

item[3] = "child";

} else if (age <= 20) {

item[3] = "teenager";

} else if (age <= 40) {

item[3] = "youth";

} else if (age <= 60) {

item[3] = "middle-aged";

} else {

item[3] = "elderly";

}

if (item[4].equals("0")) {

item[4] = "noSibsp";

} else {

item[4] = "yesSibsp";

}

if (item[5].equals("0")) {

item[5] = "noParch";

} else {

item[5] = "yesParch";

}

bw.write(item[0] + " " + item[1] + " " + item[2] + " " + item[3] + " " + item[4] + " " + item[5]+" "+item[6]);

bw.newLine(); // 添加新的数据行

}

bw.close();

1. **找出频繁项集**

首先，读取文件。

String encoding = "GBK"; // 字符编码(可解决中文乱码问题 )

File file = new File("./predata.txt");

if (file.isFile() && file.exists()) {

InputStreamReader read = new InputStreamReader(

new FileInputStream(file), encoding);//读入文件

BufferedReader bufferedReader = new BufferedReader(read);

String lineTXT = null;

while ((lineTXT = bufferedReader.readLine()) != null) {//读一行文件

String[] lineString = lineTXT.split(" "); //以空格分隔数据

List<String> lineList = new ArrayList<String>();

for (int i = 0; i < lineString.length; i++) {

lineList.add(lineString[i]);

}

record.add(lineList);//将数据加入LIstArray中

}

read.close();

}

编写函数获得频繁1-项集，设定频繁项集的最小支持度为0.2。

/\*\*

\* 首先用List<List<String>>类型的record将矩阵形式的数据读入内存；

\*/

record = getRecord();// 获取原始数据记录

List<List<String>> cItemset = findFirstCandidate();// 获取第一次的备选集

List<List<String>> lItemset = getSupportedItemset(cItemset);// 获取备选集cItemset满足支持的集合

for(int i=0;i<lItemset.size();i++){

System.out.println(lItemset.get(i));

}

抽取完后，使用println函数打印抽取的频繁1-项集情况：

**图1频繁1-项集的分布情况**

[Dead] 0.5955056179775281

[3class] 0.49859550561797755

[male] 0.6362359550561798

[youth] 0.5393258426966292

[yesSibsp] 0.34129213483146065

[noParch] 0.7289325842696629

[S] 0.7780898876404494

[Survived] 0.4044943820224719

[1class] 0.25842696629213485

[female] 0.3637640449438202

[noSibsp] 0.6587078651685393

[yesParch] 0.2710674157303371

[2class] 0.24297752808988765

频繁k-项集计算如下：

频繁2-项集如下：

/\*\*

\* 程序先求出k-1备选集，由备选集和数据库记录record求得满足支持度的k-1级集合，在满足支持度集合中求出满足自信度的集合，

\* 若满足置信度的集合为空，程序停止； 否则输出满足自信度的集合，以及对应的支持度和自信度，并由满足支持度的k-1级集合求出k级备选集，进入下一轮循环；

\* 直至程序结束，输出全部频繁集

\*/

while (endTag != true) {// 只要能继续挖掘

List<List<String>> ckItemset = getNextCandidate(lItemset);// 获取第下一次的备选集

List<List<String>> lkItemset = getSupportedItemset(ckItemset);// 获取备选集cItemset满足支持的集合

getConfidencedItemset(lkItemset, lItemset, dkCountMap, dCountMap);// 获取备选集cItemset满足置信度的集合

if (confItemset.size() != 0)// 满足置信度的集合不为空

{

printConfItemset(confItemset);// 打印满足置信度的集合

}

confItemset.clear();// 清空置信度的集合

cItemset = ckItemset;// 保存数据，为下次循环迭代准备

lItemset = lkItemset;

dCountMap.clear();

dCountMap.putAll(dkCountMap);

}

[Dead, 3class] 0.3792134831460674

[Dead, male] 0.5056179775280899

[Dead, youth] 0.3258426966292135

[Dead, noParch] 0.4705056179775281

[Dead, S] 0.4957865168539326

[Dead, noSibsp] 0.4157303370786517

[3class, male] 0.3553370786516854

[3class, youth] 0.2696629213483146

[3class, noParch] 0.3707865168539326

[3class, S] 0.40730337078651685

[3class, noSibsp] 0.3497191011235955

[male, youth] 0.3497191011235955

[male, noParch] 0.5168539325842697

[male, S] 0.5168539325842697

[male, noSibsp] 0.4592696629213483

[youth, noParch] 0.4452247191011236

[youth, S] 0.42696629213483145

[youth, Survived] 0.21348314606741572

[youth, noSibsp] 0.38202247191011235

[yesSibsp, S] 0.2598314606741573

[noParch, S] 0.5730337078651685

[noParch, Survived] 0.25842696629213485

[noParch, female] 0.21207865168539325

[noParch, noSibsp] 0.5646067415730337

[S, Survived] 0.28230337078651685

[S, female] 0.2612359550561798

[S, noSibsp] 0.5182584269662921

[S, yesParch] 0.2050561797752809

[S, 2class] 0.21910112359550563

[Survived, female] 0.273876404494382

[Survived, noSibsp] 0.24297752808988765

频繁3-项集：

[Dead, 3class, male] 0.30196629213483145

[Dead, 3class, youth] 0.20926966292134833

[Dead, 3class, noParch] 0.29213483146067415

[Dead, 3class, S] 0.32162921348314605

[Dead, 3class, noSibsp] 0.2654494382022472

[Dead, male, youth] 0.2851123595505618

[Dead, male, noParch] 0.4241573033707865

[Dead, male, S] 0.42134831460674155

[Dead, male, noSibsp] 0.375

[Dead, youth, noParch] 0.2851123595505618

[Dead, youth, S] 0.27247191011235955

[Dead, youth, noSibsp] 0.24157303370786518

[Dead, noParch, S] 0.4002808988764045

[Dead, noParch, noSibsp] 0.3848314606741573

[Dead, S, noSibsp] 0.34691011235955055

[3class, male, youth] 0.2050561797752809

[3class, male, noParch] 0.2949438202247191

[3class, male, S] 0.300561797752809

[3class, male, noSibsp] 0.2710674157303371

[3class, youth, noParch] 0.23735955056179775

[3class, youth, S] 0.22893258426966293

[3class, youth, noSibsp] 0.21207865168539325

[3class, noParch, S] 0.3089887640449438

[3class, noParch, noSibsp] 0.3160112359550562

[3class, S, noSibsp] 0.2851123595505618

[male, youth, noParch] 0.31741573033707865

[male, youth, S] 0.28230337078651685

[male, youth, noSibsp] 0.27247191011235955

[male, noParch, S] 0.4297752808988764

[male, noParch, noSibsp] 0.42696629213483145

[male, S, noSibsp] 0.37780898876404495

[youth, noParch, S] 0.35814606741573035

[youth, noParch, noSibsp] 0.34831460674157305

[youth, S, noSibsp] 0.30196629213483145

[noParch, S, noSibsp] 0.45646067415730335

频繁4-项集：

[Dead, 3class, male, noParch] 0.2542134831460674

[Dead, 3class, male, S] 0.25842696629213485

[Dead, 3class, male, noSibsp] 0.2303370786516854

[Dead, 3class, noParch, S] 0.25140449438202245

[Dead, 3class, noParch, noSibsp] 0.25

[Dead, 3class, S, noSibsp] 0.2247191011235955

[Dead, male, youth, noParch] 0.25842696629213485

[Dead, male, youth, S] 0.23735955056179775

[Dead, male, youth, noSibsp] 0.2205056179775281

[Dead, male, noParch, S] 0.36235955056179775

[Dead, male, noParch, noSibsp] 0.3553370786516854

[Dead, male, S, noSibsp] 0.31741573033707865

[Dead, youth, noParch, S] 0.24297752808988765

[Dead, youth, noParch, noSibsp] 0.22893258426966293

[Dead, youth, S, noSibsp] 0.20224719101123595

[Dead, noParch, S, noSibsp] 0.32865168539325845

[3class, male, noParch, S] 0.25280898876404495

[3class, male, noParch, noSibsp] 0.26264044943820225

[3class, male, S, noSibsp] 0.23174157303370788

[3class, youth, noParch, S] 0.20365168539325842

[3class, youth, noParch, noSibsp] 0.20224719101123595

[3class, noParch, S, noSibsp] 0.26264044943820225

[male, youth, noParch, S] 0.2612359550561798

[male, youth, noParch, noSibsp] 0.2598314606741573

[male, youth, S, noSibsp] 0.2205056179775281

[male, noParch, S, noSibsp] 0.3609550561797753

[youth, noParch, S, noSibsp] 0.2808988764044944

频繁5-项集：

[Dead, 3class, male, noParch, S] 0.2205056179775281

[Dead, 3class, male, noParch, noSibsp] 0.22612359550561797

[Dead, 3class, noParch, S, noSibsp] 0.21348314606741572

[Dead, male, youth, noParch, S] 0.21910112359550563

[Dead, male, youth, noParch, noSibsp] 0.21067415730337077

[Dead, male, noParch, S, noSibsp] 0.3061797752808989

[3class, male, noParch, S, noSibsp] 0.2247191011235955

[male, youth, noParch, S, noSibsp] 0.2148876404494382

可以看出，总共抽出了114项频繁集，其中3个变元的数目最多，有35条。

1. **导出关联规则，计算支持度和置信度**

使用Apriori算法，抽取关联规则。这里，我设置的筛选条件为支持度大于20%，置信度大于50%。

/\*\*

\* @param lkItemset

\* @param lItemset

\* @param dkCountMap2

\* @param dCountMap2 根据lkItemset，lItemset，dkCountMap2，dCountMap2求出满足自信度的集合

\*/

private static List<List<String>> getConfidencedItemset(

List<List<String>> lkItemset, List<List<String>> lItemset,

Map<Integer, Integer> dkCountMap2, Map<Integer, Integer> dCountMap2) {

for (int i = 0; i < lkItemset.size(); i++) {

getConfItem(lkItemset.get(i), lItemset, dkCountMap2.get(i),

dCountMap2);

}

return null;

}



**图2 查看抽取的关联规则**

这样，我抽出了305条关联规则。可以发现许多关联规则并不是我想要的，我更希望专注于右边的预测变元（如是否存活），因此，我需要对规则集进行筛选与评价，获得我想要的规则。

1. **去除冗余的规则**

我抽出了305条关联规则。可以发现许多关联规则并不是我想要的，我更希望专注于右边的预测变元（如是否存活）。

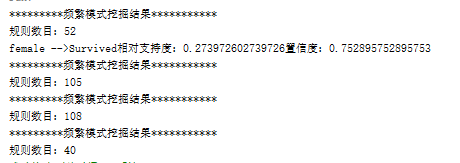
对于关联规则，首先按照变元进行选取。我想知道哪种人最可能存活，因此将右变元为存活(Survived)的规则子集筛选出来：

//筛选右变元为幸存的规则子集

temp=confItemset2.get(i).get(confItemset2.get(i).size() - 2);

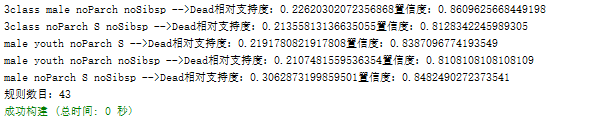
if(!temp.contains("Survived")) continue;

我可以得到图5所示结果，只有一条有价值的规则，就是，女性的存活率比较高，有27%的数据支持这个结论，其置信度达到了75%。



**图3 右变元为存活的规则**

接下来，我想分析，查看哪些人更可能在这场灾难中丧命。因此，抽取右变元为死亡(Dead)的子集。



**图4 右变元为死亡的规则数目**

可以看到，这次我抽取了43条规则，数目较多。因此，需要对这些规则进行评价筛选。

1. **关联规则评价，使用Lift指标**

在这里，我分别使用支持度，置信度与提升值对规则进行评价。

//计算lift值

private static List<List<String>> liftDeadItemset(){

List<List<String>> liftList = new ArrayList<List<String>>();

List<String> liftString = new ArrayList<String>();

double lift=0.0;

int size=0;

for(int i=0;i<deadItemset.size();i++){

liftString = deadItemset.get(i);

size = liftString.size();

lift = Double.parseDouble(liftString.get(size-1))/Dead\_Support;

liftString.add(""+lift);

liftList.add(liftString);

}

return liftList;

}



**图5 lift值**

1. **按照支持度排序**

@Override

public int compareTo(Item o) {

//按照支持度排序

// return -(this.getSupport().compareTo(o.getSupport()));

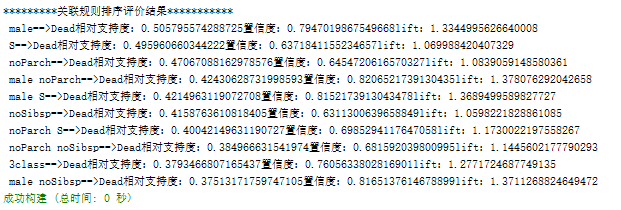
//按照置信度排序

return -(this.getConfidence().compareTo(o.getConfidence()));

//按照lift值排序

// return -(this.getLift().compareTo(o.getLift()));

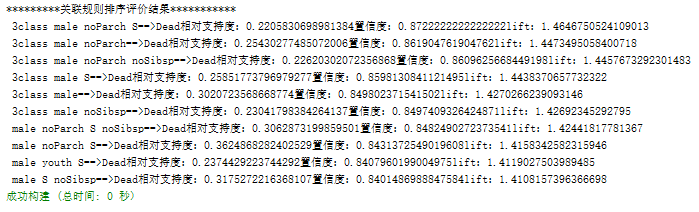
}



可以看到，其中男性出现的频率较高，并且有较大的数据支持量。

1. **按照置信度排序**

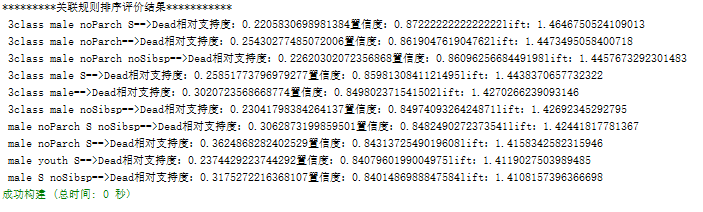
接下来，按照置信度对这些规则进行排序，获取到推测更为准确的规则。



**图6 选取置信度最高的前十条“死亡”规则**

**C)按照提升值排序**

按照提升值对这些规则进行排序，可以得到质量较高的规则。



**图7 选取置信度最高的前十条“死亡”规则**

我继续分析，可以看出，几个关键字眼出现了规则中noParch（没有父母或孩子），noSibSp（没有兄弟姐妹），男性，以及Southampton登船口。

1. **可视化**

由于matlab和java中没有可视化的包，所以采用R实现，使用扩展包arulesViz。

#加载算法库

library("Matrix")

library("arules")

library("arulesViz")

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#筛选右变元为死亡的规则子集

x=subset(rules,subset=rhs%in%"Dead")

x

inspect(x)

#画散点图

plot(x)

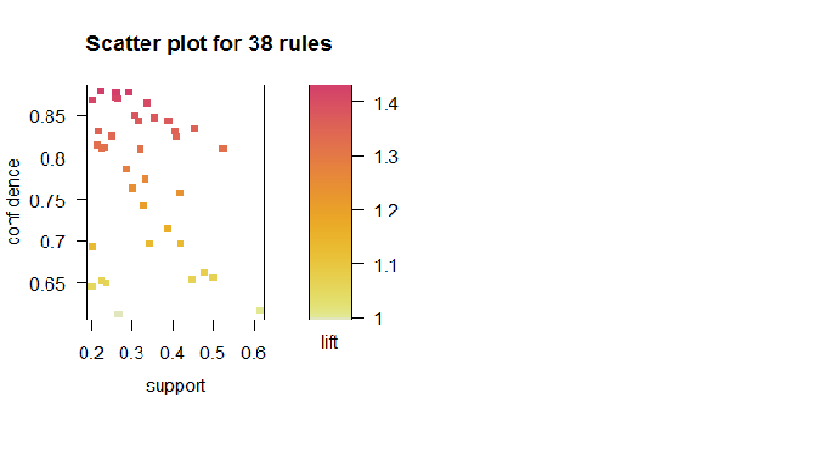
plot(x, measure = c("support", "lift"), shading = "confidence")

#画泡泡图

plot(rules, method = "grouped")

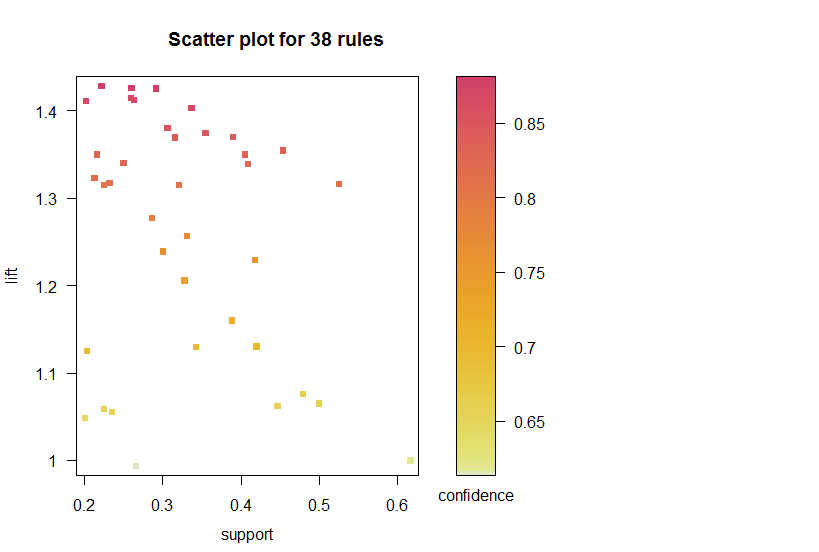
#画平行坐标图

plot(x, method="paracoord", control=list(reorder=TRUE))



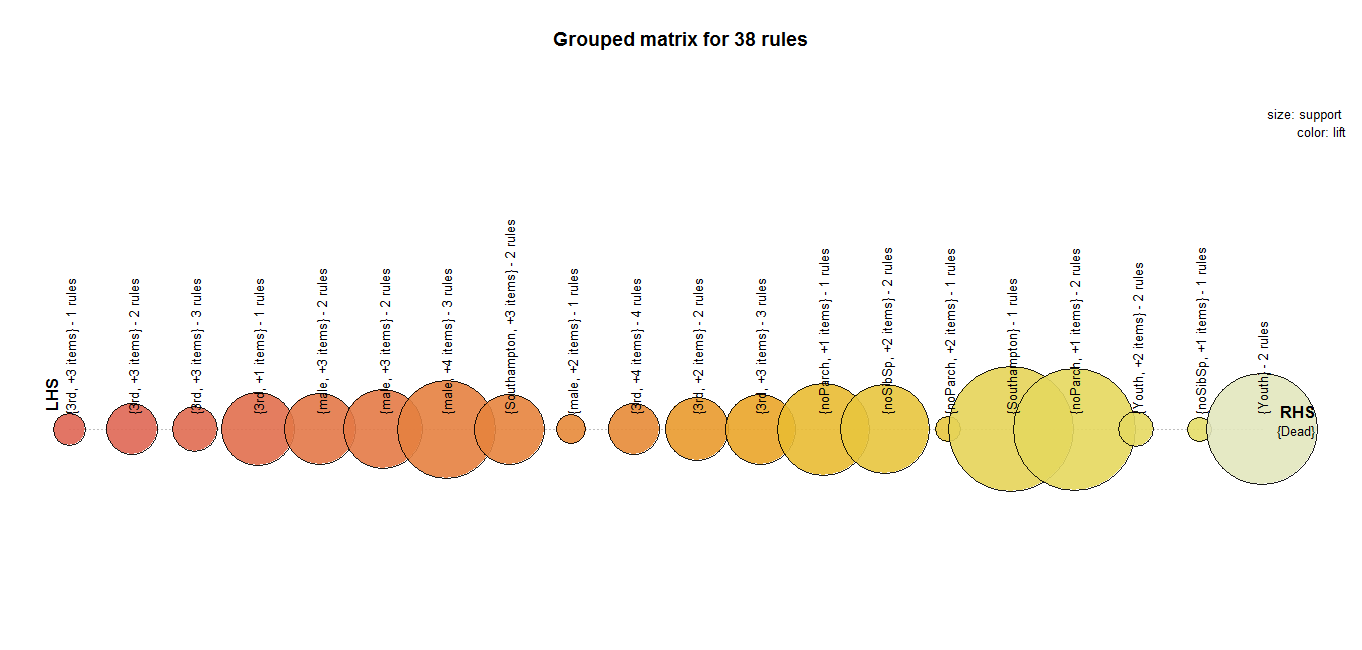
**图8 散点图**

从图中可以看出，高lift对应低support。

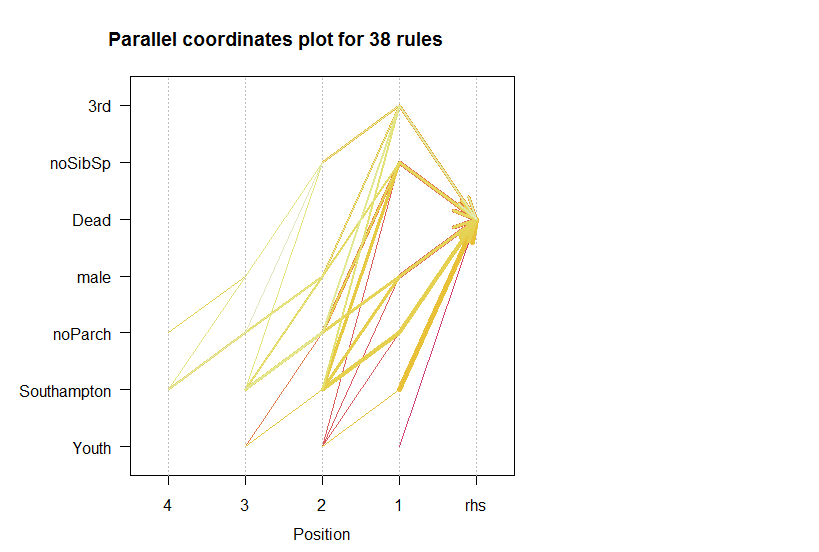


**图9 散点图**

从图中看出，很多规则都有高lift。



**图10 泡泡图**



**图11 平行坐标图**

1. **结论**

经过以上关联规则抽取，我得到初步结论，在泰坦尼克号遭遇不测时，船上优先考虑了女士的逃生路线。对于船内设计，三等座因其价格较低，在船舱的位置更不利于逃生。同时，推测最先的进水口应在Southampton口附近，导致转移时间较短，附近的人更容易死亡。在灾害发生时，无亲无故的人，求生欲望更弱，死亡率也会更高一些。