《相关性度量》算法的实现

1、算法简介

算法的介绍：

> 支持度和置信度度量不足以过滤掉无趣的关联规则，为了处理这个问题，可以使用相关性度量来扩充关联规则的支持度-置信度度量。为了更准确地挖掘出用户需要的有趣的模式，仅仅用支持度和置信度来度量已经不够，那么就需要添加一些相关性度量来判断两个项之间的相关性和无关性。

2、算法描述

> 输入：数据集D，度量方式 算法流程(伪代码):

```
function func_api(数据集，度量方式)
    获取输入数据集数量N
    for i=0:N-1
        调用要求的度量方式进行相关性计算
        将结果添加到matrix当中
    end
    返回matrix
end
```

3、参数列表

> func_api(mc=mc,func='all')

Parameters	参数介绍
`mc`	`int,optional,default:[0,0,0,0]`
`func`	`str,default:'all'`

4、方法列表

Methods	方法介绍
`func_api(mc=[0,0,0,0],func='all')`	所有度量方法的接口
`kaf(mc)`	使用卡方分析的方法进行相关性度量
`liftDegree(mc)`	使用提升度方法进行相关性度量
`totalConfidence(mc)`	使用全置信度方法进行相关性度量
`maxConfidence(mc)`	使用最大置信度方法进行相关性度量
`kulc(mc)`	使用kulc方法进行相关性度量
`cosine(mc)`	使用余弦方法进行相关性度量

5、方法参数列表

Methods	方法参数介绍	返回值
`func_api(mc=[0,0,0,0],func='all')`		
`func_api(mc=[0,0,0,0],func='all')`	`mc: 混淆矩阵;func: 调用的度量方法`	度量值矩阵
`kaf(mc)`	`mc: 混淆矩阵`	度量矩阵
`liftDegree(mc)`	`mc: 混淆矩阵`	度量矩阵
`totalConfidence(mc)`	`mc: 混淆矩阵`	度量矩阵
`maxConfidence(mc)`	`mc: 混淆矩阵`	度量矩阵
`kulc(mc)`	`mc: 混淆矩阵`	度量矩阵
`cosine(mc)`	`mc: 混淆矩阵`	度量矩阵

6、使用示例

(1) 问题描述

> 假设商店有milk和coffee销售的统计量矩阵如下表所示，我们的目的是通过这些数据，度量变量之间的相关性，从而挖掘出一些有趣的信息

-	Milk	No-Milk	和
Coffee	M&&C	No-M&&C	C
No-Coffee	M&&No-C	No-M&&No-C	No-C
和	M	No-M	10000

(2) 数据集

> 测试数据集为一个6*4的mc矩阵，数据为四维的，共6个样本。

数据集	M&&C	No-M&&C	M&&No-C	No-M&&No-C
D1	10000	1000	1000	100000
D2	10000	1000	1000	100
D3	100	1000	1000	100000
D4	1000	1000	1000	100000
D5	1000	100	10000	100000
D6	1000	10	100000	100000

(3) 实验设置

> 直接对上述的6条记录进行相关性度量，func='all':用全部的6种度量分别进行计算

(4) 应用程序

```
#datamining.py
import numpy as np
def kaf(mc):
    sumItems=np.sum(mc);
    alphaMilk=(mc[0]+mc[2])/sumItems;
    alphaCoff=(mc[0]+mc[1])/sumItems;

    value=[(mc[0]-alphaMilk*alphaCoff*sumItems)**2/(alphaMilk*alphaCoff*sumItems),
           (mc[1]-(1-alphaMilk)*alphaCoff*sumItems)**2/((1-
alphaMilk)*alphaCoff*sumItems),
           (mc[2]-alphaMilk*(1-alphaCoff)*sumItems)**2/(alphaMilk*(1-
alphaCoff)*sumItems),
           (mc[3]-(1-alphaMilk)*(1-alphaCoff)*sumItems)**2/((1-alphaMilk)*(1-
alphaCoff)*sumItems)]
    return value

def liftDegree(mc):
    sumItems = np.sum(mc);
    alphaMilk = (mc[0] + mc[2]) / sumItems;
    alphaCoff = (mc[0] + mc[1]) / sumItems;
    alphaMilkCoff=mc[0]/sumItems;

    return alphaMilkCoff/(alphaMilk*alphaCoff)

def totalConfidence(mc):
    sumItems = np.sum(mc);
    alphaMilk = (mc[0] + mc[2]) / sumItems;
    alphaCoff = (mc[0] + mc[1]) / sumItems;
    alphaMilkCoff = mc[0] / sumItems;

    return alphaMilkCoff/max(alphaMilk,alphaCoff);
```

```
def maxConfidence(mc):
    sumItems = np.sum(mc);
    alphaMilk = (mc[0] + mc[2]) / sumItems;
    alphaCoff = (mc[0] + mc[1]) / sumItems;
    alphaMilkCoff = mc[0] / sumItems;

    return alphaMilkCoff / min(alphaMilk, alphaCoff);

def kulc(mc):
    sumItems = np.sum(mc);
    alphaMilk = (mc[0] + mc[2]) / sumItems;
    alphaCoff = (mc[0] + mc[1]) / sumItems;
    alphaMilkCoff = mc[0] / sumItems;

    return 1/2*(alphaMilkCoff/alphaMilk+alphaMilkCoff/alphaCoff)

def cosine(mc):
    sumItems = np.sum(mc);
    alphaMilk = (mc[0] + mc[2]) / sumItems;
    alphaCoff = (mc[0] + mc[1]) / sumItems;
    alphaMilkCoff = mc[0] / sumItems;
    return alphaMilkCoff/(alphaMilk*alphaCoff)**0.5

def func_api(mc=[0,0,0,0],func='all'):
    matrix = [];

    if func=='all':
        for i in range(len(mc)):

 matrix.append([np.sum(kaf(mc[i])),liftDegree(mc[i]),totalConfidence(mc[i]),\
                           maxConfidence(mc[i]),kulc(mc[i]),cosine(mc[i])]);
    elif func=='kaf':
        for i in range(len(mc)):
            matrix.append([np.sum(kaf(mc[i]))]);
    elif func=='liftDegree':
        for i in range(len(mc)):
            matrix.append([liftDegree(mc[i])]);
    elif func=='totalConfidence':
        for i in range(len(mc)):
            matrix.append([totalConfidence(mc[i])]);
    elif func=='maxConfidence':
        for i in range(len(mc)):
            matrix.append([maxConfidence(mc[i])]);
    elif func=='kulc':
        for i in range(len(mc)):
            matrix.append([kulc(mc[i])]);
    elif func=='consine':
        for i in range(len(mc)):
            matrix.append([cosine(mc[i])]);
    else:
        return -1;

    return matrix;
```

```
if __name__ == '__main__':
    mc = [];
    mc.append([10000, 1000, 1000, 100000]);#m_c nm_c m_nc nm_nc
    mc.append([10000, 1000, 1000, 100]);
    mc.append([100, 1000, 1000, 100000]);
    mc.append([1000, 1000, 1000, 100000]);
    mc.append([1000, 100, 10000, 100000]);
    mc.append([1000, 10, 100000, 100000]);

    print(["mileANDcoffee","NOTmilkANDcoffee","milkNOTcoffee","NOTmileNOTcoffee"])
    for i in range(len(mc)):
        print(mc[i])

    func='all'
    matrix=func_api(mc,func);
    np.set_printoptions(formatter={'float':'{:10.3f}'.format});
    # np.set_printoptions(precision=2,suppress=True)
    if func=='all':
        print(["kaf",'liftDegree','totalConfidence','maxConfidence','kulc','consine'])
    else:
        print([func])
    print(np.array(matrix))
```

(5) 运行结果

程序运行之后，对6组数据分别运用6种度量方法，给出了各自的相关性度量结果 如下图所示

```
['mileANDcoffee', 'NOTmilkANDcoffee', 'milkNOTcoffee', 'NOTmileNOTcoffee']
[10000, 1000, 1000, 100000]
[10000, 1000, 1000, 100]
[100, 1000, 1000, 100000]
[1000, 1000, 1000, 100000]
[1000, 100, 10000, 100000]
[1000, 10, 100000, 100000]
['kaf', 'liftDegree', 'totalConfidence', 'maxConfidence', 'kulc', 'consine']
[[ 90556.761      9.256      0.909      0.909      0.909      0.909]
 [     0.000      1.000      0.909      0.909      0.909      0.909]
 [   670.012      8.438      0.091      0.091      0.091      0.091]
 [ 24740.295     25.750      0.500      0.500      0.500      0.500]
 [  8172.827      9.182      0.091      0.909      0.500      0.287]
 [   965.544      1.970      0.010      0.990      0.500      0.099]]
```