韩林洁　2620170052

数据挖掘大作业二：关联规则挖掘

问题描述：

1. 数据源

从大作业一的两个数据集中任选一个进行分析。

2. 要求

对数据集进行处理，转换成适合关联规则挖掘的形式；

找出频繁项集；

导出关联规则，计算其支持度和置信度

对规则进行评价，可使用 Lift，也可以使用教材中所提及的其它指标

3. 提交的内容

对数据集进行处理的源程序

关联规则挖掘的源程序

挖掘结果及分析

挖掘过程的报告

问题解答：

对数据集 Permit_Building 进行关联规则分析，针对 Proposed Units，Estimated Cost，Revised Cost ，Number of Proposed Stories，Number of Existing Stories 这五个数值属性进行关联规则挖掘

一、数据的预处理

1、读取文件

```
db='Building_Permits.csv'
#读取 csv 文件，生成 data frame
data=pd.read_csv(db,low_memory=False)
```

2、查看数据的摘要，重点关注缺失值

```
explore=attri.describe(percentiles=[],include='all').T
explore['null']=len(attri)-explore['count']
explore=explore[['null','max','min']]
```

```
explore.columns=[u'空值数',u'最大值',u'最小值']
explore.to_csv(result_file)
```

结果在 result_file.csv 文件中，结果如下：

2、缺失值填充

用众数填充属性的缺失值

```
# 通过众数来填充缺失值
for i in range(0,5):
    MostFrequentElement = attri.iloc[:,[i]].apply(pd.value_counts).idxmax()
    attri.iloc[:, [i]] = attri.iloc[:,[i]].fillna(value=MostFrequentElement)
    print('success')
    print(attri.info())
```

3、数据变换（包括数据的规范化和离散化）

（1)数据的规范化，用最小-最大规范化

```
data=(data-data.min(axis=0))/(data.max(axis=0)-data.min(axis=0))
```

（2)数据的离散化，

由于 apriori 算法无法处理连续型数值变量，所以需要离散化，用 k-meas 聚类进行数据离散化，调用 sklearn.cluster 的 Kmeans 库。数据离散化代码如下，

```
for i in range(len(keys)):
#调用 k-means 算法，进行聚类离散化
print(u'正在进行“%s”的聚类...' % keys[i])
kmodel = KMeans(n_clusters = k) #n_jobs 是并行数，一般等于 CPU 数较好
kmodel.fit(data[[keys[i]]].as_matrix()) #训练模型
r1 = pd.DataFrame(kmodel.cluster_centers_, columns = [typelabel[keys[i]]]) #聚类中心
r2 = pd.Series(kmodel.labels_).value_counts() #分类统计
#记录各个类别的数目
r2 = pd.DataFrame(r2, columns = [typelabel[keys[i]]+'n']) #转为 DataFrame,
print(pd.concat([r1, r2], axis = 1))
print(typelabel[keys[i]])
r = pd.concat([r1, r2], axis = 1).sort_values(typelabel[keys[i]]) #匹配聚类中心和类别数目
r.index = [1, 2, 3, 4, 5, 6, 7, 8] #8 个类别
r[typelabel[keys[i]]] = pd.rolling_mean(r[typelabel[keys[i]]], 2) #rolling_mean()用来计算相邻 2
列的均值，以此作为边界点。
r[typelabel[keys[i]]][1] = 0.0 #这两句代码将原来的聚类中心改为边界点。
result = result.append(r.T)
result = result.sort_index() #以 Index 排序，即以 A,B,C,D 顺序排
result.to_csv(processedfile)
```

结果输出到 processedfile 文件，结果如下：

	1	2	3	4	5	6	7	8
A	0	0.017558	0.056141	0.115611	0.187667	0.256222	0.329556	0.684667

An	957	22	6	5	4	3	1	2
B	0	0.010041	0.039134	0.095634	0.193168	0.378713	0.675033	0.923583
Bn	893	79	14	6	4	2	1	1
C	0	0.031758	0.087001	0.203029	0.350728	0.511859	0.655449	0.825
Cn	685	225	34	22	15	6	8	5
D	0	0.031747	0.087521	0.203347	0.35232	0.513622	0.655449	0.825
Dn	685	220	37	23	16	6	8	5

其中数字代表类别，A,B,C,D 代表不同属性，An,Bn,Cn,Dn 代表数量，即 A 属性中（0,0.017558）有 957 个，B 属性中（0,0,10041）有 893 个，依此类推.

(3)把得到的聚类结果进行变换，便于运行 apriori 算法

代码如下：

```
aprioriData=pd.DataFrame()
for i in range(len(keys)):
    N=typelabel[keys[i]] #A,B,C,D
    C=keys[i]
    bool1 = result.loc[N, 1] <= data[C]
    bool2 = result.loc[N, 2] <= data[C]
    bool3 = result.loc[N, 3] <= data[C]
    bool4 = result.loc[N, 4] <= data[C]
    bool5 = result.loc[N, 5] <= data[C]
    bool6 = result.loc[N, 6] <= data[C]
    bool7 = result.loc[N, 7] <= data[C]
    bool8 = result.loc[N, 8] <= data[C]
    typeN=1*(bool1 & ~bool2)+2*(bool2 & ~bool3)+3*(bool3 &~bool4)+4*(bool4
&~bool5)+5*(bool5&~bool6)+6*(bool6&~bool7)+7*(bool7&~bool8)+8*(bool8)
    typeN=typeN.replace({1:N+'1',2:N+'2',3:N+'3',4:N+'4',5:N+'5',6:N+'6',7:N+'7',8:N+'8'})
    aprioriData=pd.concat([aprioriData,typeN],axis=1)
aprioriData.to_csv(apriori)
```

变换后的数据输入到 apriori.csv 中，如下（只截取了前 10 条数据）：

	Proposed Units	Revised Cost	Number of Proposed Stories	Number of Existing Stories
0	A1	B1	C1	D3
1	A1	B1	C1	D3
2	A2	B1	C3	D3
3	A1	B1	C1	D1
4	A1	B2	C1	D2
5	A5	B1	C2	D2
6	A1	B1	C2	D2
7	A1	B1	C1	D1
8	A1	B1	C1	D1
9	A1	B1	C1	D1

10 A1 B1 C1 D1

二、aprior 算法进行关联规则挖掘

1. 在 apriori.py 中依次定义了两个函数：

```
#自定义连接函数，用于实现 L_{k-1}到 C_k 的连接
def connect_string(x, ms):
#寻找关联规则的函数
def find_rule(d, support, confidence, ms = u'--'):
```

2. Aprior_rules 调用 apriori 中的函数，来实现关联规则的挖掘，代码如下：

```
inputfile = 'apriori.csv' #输入事务集文件
data = pd.read_csv(inputfile, header=None, dtype = object)
start = time.clock() #计时开始
print(u'\n 转换原始数据至 0-1 矩阵...')
ct = lambda x : pd.Series(1, index = x) #转换 0-1 矩阵的过渡函数
b = map(ct, data.as_matrix()) #用 map 方式执行
data = pd.DataFrame(list(b)).fillna(0) #实现矩阵转换，空值用 0 填充
end = time.clock() #计时结束
print(u'\n 转换完毕，用时：%0.2f 秒' %(end-start))
del b #删除中间变量 b，节省内存
support = 0.6 #最小支持度
confidence = 0.75 #最小置信度
ms = '---' #连接符，默认'--'，用来区分不同元素，如 A--B。需要保证原始表格中不含有该字符
start = time.clock() #计时开始
print(u'\n 开始搜索关联规则...')
find_rule(data, support, confidence, ms)
end = time.clock() #计时结束
print(u'\n 搜索完成，用时：%0.2f 秒' %(end-start))
```

得到的结果如下：

结果为：

	support	confidence
C1---A1	0.684316	1.000000
C1---D1---A1	0.673327	1.000000
B1---C1---A1	0.645355	1.000000
B1---C1---D1---A1	0.636364	1.000000
D1---A1	0.681319	0.995620
B1---D1---A1	0.640360	0.995342
A1---B1---D1---C1	0.636364	0.993760
B1---D1---C1	0.636364	0.989130
A1---D1---C1	0.673327	0.988270
B1---C1---D1	0.636364	0.986068
A1---B1---C1---D1	0.636364	0.986068
D1---C1	0.673327	0.983942
C1---D1	0.673327	0.983942
A1---C1---D1	0.673327	0.983942
B1---D2---A1	0.178821	0.983516
B1---D2---C2	0.178821	0.983516
B1---C2---D2---A1	0.175824	0.983240

从图中可以看出 C1—A1，C1—D1—A1 等上述规则均满足支持度大于等于 0.6，置信度大于等于 0.7，均为强关联规则，对于 A1—C1 规则的解释如下：

Proposed Unit 中（0.001758）的值和 Estimated Cost 的（0，0.0311758）值同时发生的概率是 0.684316，Proposed Unit 中（0.001758）值出现的情况下，Estimated Cost 的（0，0.0311758）值出现的概率是 1，即必然出现。

二、关联规则结果的评估

用提升度来评价关联规则