北京理工大學

研究生课程报告

频繁模式与关联规则挖掘

学　　院：	计算机学院
专　　业：	电子信息
指导教师：	汤世平
姓　　名：	李云彤
学　　号：	3220200910

2021 年 5 月 31 日

一、实验要求

- 对数据集进行处理，转换成适合进行关联规则挖掘的形式；
- 找出频繁模式；
- 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、卡方和其它教材中提及的指标, 至少 2 种；
- 对挖掘结果进行分析；

  可视化展示。

二、数据集选择

选择进行频繁模式与关联规则挖掘的数据集为：Wine Reviews 数据集。Wine Reviews 数据集为全球葡萄酒信息的的数据集，其中标称属性包括国家(country)、省份(province)、 地区 1(region_1)、地区 2（region_2）、品种（variety）、酒厂（winery），数值属性包括 points 和 price。首先查看数据的大体信息：

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 280901 entries, 0 to 280900
Data columns (total 13 columns):
 #   Column                 Non-Null Count   Dtype
---  ------                 --------------   -----
 0   country                280833 non-null  object
 1   description            280901 non-null  object
 2   designation            197701 non-null  object
 3   points                 280901 non-null  int64
 4   price                  258210 non-null  float64
 5   province               280833 non-null  object
 6   region_1               234594 non-null  object
 7   region_2               111464 non-null  object
 8   variety                280900 non-null  object
 9   winery                 280901 non-null  object
 10  taster_name            103727 non-null  object
 11  taster_twitter_handle  98758 non-null   object
 12  title                  129971 non-null  object
dtypes: float64(1), int64(1), object(11)
memory usage: 27.9+ MB
```

接下来对数据集进行预处理，删除掉数据集中的缺失值，缺失值统计信息如下图所示：

```
Out[40]: country                   68
         description                0
         designation            83200
         points                     0
         price                  22691
         province                  68
         region_1                46307
         region_2               169437
         variety                     1
         winery                      0
         taster_name            177174
         taster_twitter_handle  182143
         title                  150930
         dtype: int64
```

最后选取适合进行频繁模式和关联规则挖掘的属性:country、points、price、provin、region_1、region_2、variety 以及 winery，将预处理后的数据集中的这些属性字段提出出来，并放在一个列表里，作为后续频繁模式挖掘和关联规则挖掘的数据对象。

三、频繁模式挖掘

频繁模式是数据集中频繁出现的项集、序列或子结构。如在购物车分析中，会分析哪些商品频繁的被客户同时购买，在网页日志分析中，会分析用户在浏览“手机”页面后，通常会继续浏览哪些页面等，包含着一些重要的信息。由于采用暴力的方法进行频繁模式挖掘需要指数级的时间复杂度，因此需要采用一些频繁模式挖掘的算法对其进行优化，本次实验中采用的算法为 Apriori 算法。

Apriori 算法：算法的核心思想是如果一个集合是频繁项集，则它的所有子集都是频繁项集；如果一个集合不是频繁项集，则它的所有超集都不是频繁项集。因此可以根据这两条规则进行剪枝，避免指数级的时间复杂度。

算法流程如下：

1）找到频繁的一维项集 L1

2）从频繁的 Lk 维项集生成 k+1 维项集 Ck+1

3）找到 Ck+1 中的频繁项集 Lk+1

4）k=k+1，循环执行 2）-3）直至 k+1=n，n 为最大项集

5）输出各个维度的频繁项集

算法的伪代码如下图所示：

C_k: Candidate itemset of size k

F_k : Frequent itemset of size k

K := 1;
F_k := {frequent items}; // frequent 1-itemset
While (F_k != ∅) **do {** // when F_k is non-empty
C_{k+1} := candidates generated from F_k; // candidate generation
Derive F_{k+1} by counting candidates in C_{k+1} with respect to *TDB* at minsup;
k := k + 1
}
return $\cup_k F_k$ // return F_k generated at each level

为实现 Apriori 算法，首先要构建全部可能的单元素候选项集合，以列表的形式存储这些单元素候选项集合，每个单元素候选项包括属性名，属性取值。

实现代码如下：

```
def Create_C1(dataset):#频繁模式挖掘
    C1 = []
    progress = ProgressBar()
    for data in progress(dataset):
        for item in data:
            if [item] not in C1:
                C1.append([item])
    print("Created success")
    return [frozenset(item) for item in C1]
```

下面需要过滤掉小于最小支持度的非频繁项集，判断每个候选项集中元素的出现频率，过滤掉出现频率低于最小支持度的项集，最小支持度设置为 0.1。

实现代码如下：

```
def scan_D(dataset, Ck):
    Ck_count = dict()
    for data in dataset:
        for cand in Ck:
            if cand.issubset(data):
                if cand not in Ck_count:
                    Ck_count[cand] = 1
                else:
                    Ck_count[cand] += 1

    num_items = float(len(dataset))
    return_list = []
    support_data = dict()
        # 过滤非频繁项集
    for key in Ck_count:
        support  = Ck_count[key] / num_items
        if support >= 0.1:
             return_list.insert(0, key)
        support_data[key] = support
    return return_list, support_data
```

接下来要对每个候选集进行合并和筛选：

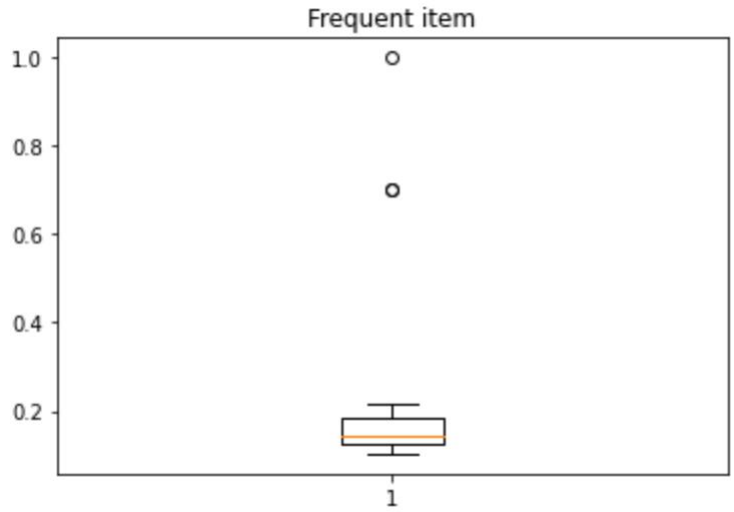
```
def apriori_gen(Lk, k):
        return_list = []
        len_Lk = len(Lk)
        for i in range(len_Lk):
            for j in range(i+1, len_Lk):
                L1 = list(Lk[i])[:k-2]
                L2 = list(Lk[j])[:k-2]
                L1.sort()
                L2.sort()
                if L1 == L2:
                    return_list.append(Lk[i] | Lk[j])
        return return_list
```

得到挖掘结果，按支持度从大到小排序，可以看到 country 字段和 US 永远同时出现，原因可能是在数据预处理中删除了大量包含空值的元组，导致剩下的数据中 country 属性只剩下了美国，同时加利福尼亚在省份中出现的频率最高，表示数据中大部分就出自加利福尼亚州：

```
In  [46]: freq
```

```
Out[46]: [{'set': [['country', 'US']], 'sup': 1.0},
          {'set': [['province', 'California']], 'sup': 0.7036379689358175},
          {'set': [['province', 'California'], ['country', 'US']],
           'sup': 0.7036379689358175},
          {'set': [['region_2', 'Central Coast']], 'sup': 0.21588165339291507},
          {'set': [['region_2', 'Central Coast'], ['province', 'California']],
           'sup': 0.21588165339291507},
          {'set': [['region_2', 'Central Coast'], ['country', 'US']],
           'sup': 0.21588165339291507},
          {'set': [['region_2', 'Central Coast'],
            ['country', 'US'],
            ['province', 'California']],
           'sup': 0.21588165339291507},
          {'set': [['region_2', 'Sonoma']], 'sup': 0.18219575480197484},
          {'set': [['province', 'California'], ['region_2', 'Sonoma']],
           'sup': 0.18219575480197484},
          {'set': [['country', 'US'], ['region_2', 'Sonoma']],
           'sup': 0.18219575480197484},
          {'set': [['province', 'California'],
            ['country', 'US'],
            ['region_2', 'Sonoma']],
           'sup': 0.18219575480197484},
```

最后进行可视化处理，以盒图的形式进行显示：



四、关联规则挖掘

关联规则是关联规则就是描述这种在一个事务中物品之间同时出现的规律的知识模式。更确切的说，关联规则通过量化的数字描述物品甲的出现对物品乙的出现有多大的影响。对于使用 Apriori 算法进行频繁模式挖掘所得到的频繁项集计算其关联规则并用评价指标进行评价。本实验利用 Lift 和 Jaccard 两种指标进行评价。

基于 Apriori 算法，首先从一个频繁项集开始，创建一个规则列表，规则右部只包含一个元素，然后对这些规则进行测试，接下来合并所有的剩余规则列表来形成一个新的规则列表，其中规则右部包含两个元素：

```
def generate_rules(L, support_data):
        big_rules_list = []
        for i in range(1, len(L)):
            for freq_set in L[i]:
                H1 = [frozenset([item]) for item in freq_set]
                if i > 1:
                    rules_from_conseq(freq_set, H1, support_data, big_rules_list)
                else:
                    cal_conf(freq_set, H1, support_data, big_rules_list)
        return big_rules_list
```

将频繁项集的元素映射到规则右部的元素列表：

```
def rules_from_conseq(freq_set, H, support_data, big_rules_list):
        m = len(H[0])
        if len(freq_set) > (m+1):
            Hmp1 = apriori_gen(H, m+1)
            Hmp1 = cal_conf(freq_set, Hmp1, support_data, big_rules_list)
            if len(Hmp1) > 1:
                rules_from_conseq(freq_set, Hmp1, support_data, big_rules_list)
```

评价指标的计算公式为：

Lift 评价指标：

$$lift(B,C)=\frac{c(B\rightarrow C)}{s(C)}=\frac{s(B\cup C)}{s(B)\times s(C)}$$

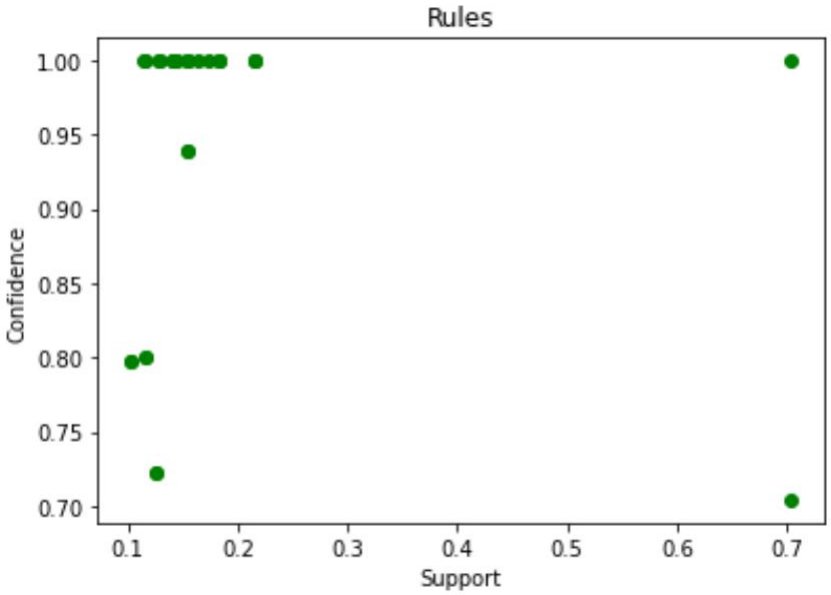
Jaccard 评价指标：

$$\text{Jaccard}(X\rightarrow Y)=\frac{Sup(X\cup Y)}{Sup(X)+Sup(Y)-Sup(X\cup Y)}$$

利用评价指标评估生成的规则：

```
def cal_conf(freq_set, H, support_data, big_rules_list):
        prunedH = []
        for conseq in H:
            sup = support_data[freq_set]
            conf = sup / support_data[freq_set - conseq]
            lift = conf / support_data[freq_set - conseq]
            jaccard = sup / (support_data[freq_set - conseq] + support_data[conseq] - sup)
            if conf >= 0.5:
                big_rules_list.append((freq_set-conseq, conseq, sup, conf, lift,jaccard))
                prunedH.append(conseq)
        return prunedH
```

根据结果绘制散点图如下：



将得到的关联规则以及评价结果按置信度排序，由于 country 字段的取值全部为 US，所以对齐分析没有实际意义，因此跳过 country 字段，通过结果可以

看出 Region_2 和 province 的关联度较高，表示该地区很可能在对应的省内：


```
 'lift': 6.472069970845482,
 'jaccard': 0.1545100724350427},
{'X_set': [['region_2', 'Columbia Valley']],
 'Y_set': [['province', 'Washington']],
 'sup': 0.1545100724350427,
 'conf': 1.0,
 'lift': 6.472069970845482,
 'jaccard': 0.9389542841500136},
{'X_set': [['region_2', 'Central Coast']],
 'Y_set': [['country', 'US']],
 'sup': 0.21588165339291507,
 'conf': 1.0,
 'lift': 4.632167598697938,
 'jaccard': 0.21588165339291507},
{'X_set': [['region_2', 'Central Coast']],
 'Y_set': [['province', 'California']],
 'sup': 0.21588165339291507,
 'conf': 1.0,
```