北京理工大学
BEIJING INSTITUTE OF TECHNOLOGY

数据挖掘
互评作业二

题　　目：频繁模式与关联规则挖掘

学　　院：计算机学院

专业名称：电子信息（计算机技术）

学　　号：3220211085

姓　　名：吴凌云

任课教师：汤世平老师

2021 年 4 月 10 日星期日

1. Oakland Crime Statistics 2011 to 2016 数据集说明及处理

在这个数据集中，一共包含 6 个数据子集，分别为 2011-2016 年度的奥克兰犯罪情况。其详细属性如下：

```
2011数据集有以下属性 Index(['Agency', 'Create Time', 'Location', 'Area Id', 'Beat', 'Priority',
       'Incident Type Id', 'Incident Type Description', 'Event Number',
       'Closed Time'],
      dtype='object')
2012数据集有以下属性 Index(['Agency', 'Create Time', 'Area Id', 'Beat', 'Priority',
       'Incident Type Id', 'Incident Type Description', 'Event Number',
       'Closed Time', 'Location 1', 'Zip Codes'],
      dtype='object')
2013数据集有以下属性 Index(['Agency', 'Create Time', 'Location ', 'Area Id', 'Beat', 'Priority',
       'Incident Type Id', 'Incident Type Description', 'Event Number',
       'Closed Time'],
      dtype='object')
2014数据集有以下属性 Index(['Agency', 'Create Time', 'Area Id', 'Beat', 'Priority',
       'Incident Type Id', 'Incident Type Description', 'Event Number',
       'Closed Time', 'Location 1', 'Zip Codes'],
      dtype='object')
2015数据集有以下属性 Index(['Agency', 'Create Time', 'Location', 'Area Id', 'Beat', 'Priority',
       'Incident Type Id', 'Incident Type Description', 'Event Number',
       'Closed Time'],
      dtype='object')
2016数据集有以下属性 Index(['Agency', 'Create Time', 'Location', 'Area Id', 'Beat', 'Priority',
       'Incident Type Id', 'Incident Type Description', 'Event Number',
       'Closed Time'],
      dtype='object')
```

通过观察数据集可得知，这六年的数据属性基本一样，值得进行分析与预处理的有如下几个属性：Agency, Location, Area id, Beat, Incident Type id，Incident Type Descripe, Event Number，其中 2012 年和 2014 年的属性为 Location 1，经过特殊处理变为 Location。由于 Incident Type id 与 Incident Type Descripe 一一对应，我们只对 Incident Type id 进行分析。Event Number 对应每一行数据，不具备重复性，不对其进行分析。

发现有部分数据存在缺失值的情况。使用上次预处理的方法，舍

去有缺失值的行后，由原来的 1046388 条数据变为剩下 859898 条数据，在此基础上进行实验。

2. 找出频繁模式

2.1 使用算法及简单介绍

算法名称：Apriori 算法

在本实验中，使用 Apriori 算法来构建频繁项集。在本实验中，我们约定支持度的阈值为 10%，置信度的阈值为 50%。

```
min_sup = 0.1 #支持度阈值
min_conf = 0.5 #置信度阈值
```

相应代码为：

Apriori 主函数：

```
def apriori(self, dataset):             #算法主体
    C1 = self.C1_generation(dataset)        #生成单元素候选项集
    dataset = [set(data) for data in dataset]
    F1, sup_rata = self.Ck_low_support_filtering(dataset, C1)
    F = [F1]
    k = 2
    while len(F[k-2]) > 0:
        Ck = self.apriori_gen(F[k-2], k)        #当候选项元素太于2时，合并时检测是否子项集满足频繁
        Fk, support_k = self.Ck_low_support_filtering(dataset, Ck)      #过滤支持度低于阈值的项集
        sup_rata.update(support_k)
        F.append(Fk)
        k += 1
    return F, sup_rata
```

单元素候选集生成函数：

```python
def C1_generation(self, dataset):        #生成单元素候选项集
    C1 = []
    progress = ProgressBar()
    for data in progress(dataset):
        for item in data:
            if [item] not in C1:
                C1.append([item])
    return [frozenset(item) for item in C1]
```

过滤低支持度函数：

```python
def Ck_low_support_filtering(self, dataset, Ck):        #过滤支持度低于阈值的项集
    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 = []
    sup_rata = dict()
    # 过滤非频繁项集
    for key in Ck_count:
        support = Ck_count[key] / num_items
        if support >= self.min_sup:
            return_list.insert(0, key)
        sup_rata[key] = support
    return return_list, sup_rata
```

合并筛选函数：

```python
def apriori_gen(self, Fk, k):        #当候选项元素太于2时，合并时检测是否子项集满足频繁
    return_list = []
    len_Fk = len(Fk)

    for i in range(len_Fk):
        for j in range(i+1, len_Fk):
            # 第k-2个项相同时，将两个集合合并
            F1 = list(Fk[i])[:k-2]
            F2 = list(Fk[j])[:k-2]
            F1.sort()
            F2.sort()
            if F1 == F2:
                return_list.append(Fk[i] | Fk[j])
    return return_list
```

最终产生的频繁项集结果保存在“频繁项集.json”文件中，效果如下图所示：


```
{"set": [["Agency", "OP"]], "sup": 1.0}
{"set": [["Priority", 2.0]], "sup": 0.80188}
{"set": [["Priority", 2.0], ["Agency", "OP"]], "sup": 0.80188}
{"set": [["Area Id", 1.0]], "sup": 0.36544}
{"set": [["Area Id", 1.0], ["Agency", "OP"]], "sup": 0.36544}
{"set": [["Area Id", 3.0]], "sup": 0.32649}
{"set": [["Agency", "OP"], ["Area Id", 3.0]], "sup": 0.32649}
{"set": [["Area Id", 2.0]], "sup": 0.30807}
{"set": [["Area Id", 2.0], ["Agency", "OP"]], "sup": 0.30807}
{"set": [["Priority", 2.0], ["Area Id", 1.0]], "sup": 0.29778}
{"set": [["Priority", 2.0], ["Area Id", 1.0], ["Agency", "OP"]], "sup": 0.29778}
{"set": [["Priority", 2.0], ["Area Id", 3.0]], "sup": 0.2557}
{"set": [["Priority", 2.0], ["Agency", "OP"], ["Area Id", 3.0]], "sup": 0.2557}
{"set": [["Priority", 2.0], ["Area Id", 2.0]], "sup": 0.2484}
{"set": [["Priority", 2.0], ["Area Id", 2.0], ["Agency", "OP"]], "sup": 0.2484}
{"set": [["Priority", 1.0]], "sup": 0.19811}
{"set": [["Priority", 1.0], ["Agency", "OP"]], "sup": 0.19811}
```


3. 导出关联规则（包含评价）

基于 2 中使用 Apriori 算法得出的频繁项集，我们计算关联规则以及使用评价指标来评价它们。本实验使用的是课件中的 Lift 和 Jaccard 两种指标进行评价。

其中计算的公式为：

支持度：

$$Sup(X) = \frac{count(X)}{all\_data}$$

置信度：

$$conf(X \rightarrow Y) = \frac{Sup(X \cup Y)}{Sup(X)}$$

Lift:

$$left(X \rightarrow Y) = \frac{Sup(X \cup Y)}{Sup(X) * Sup(Y)}$$

Jaccard:

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

以上计算对应的代码为：

```
def cal_conf(self, freq_set, H, sup_rata, strong_rules_list):          #评估规则
    prunedH = []
    for reasoned_item in H:
        sup = sup_rata[freq_set]
        conf = sup / sup_rata[freq_set - reasoned_item]
        lift = conf / sup_rata[reasoned_item]
        jaccard = sup / (sup_rata[freq_set - reasoned_item] + sup_rata[reasoned_item] - sup)
        if conf >= self.min_conf:
            strong_rules_list.append((freq_set-reasoned_item, reasoned_item, sup, conf, lift, jaccard))
            prunedH.append(reasoned_item)
    return prunedH
```

计算并保留达到置信度阈值的函数为：

```
def generate_rules(self, F, sup_rata):
    """
    产生强关联规则算法实现
    基于Apriori算法，首先从一个频繁项集开始，接着创建一个规则列表，
    其中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创建一个新的规则列表，
    其中规则右部包含两个元素。这种方法称作分级法。
    :param F: 频繁项集
    :param sup_rata: 频繁项集对应的支持度
    :return: 强关联规则列表
    """
    strong_rules_list = []
    for i in range(1, len(F)):
        for freq_set in F[i]:
            H1 = [frozenset([item]) for item in freq_set]
            # 只获取有两个或更多元素的集合
            if i > 1:
                self.rules_from_reasoned_item(freq_set, H1, sup_rata, strong_rules_list)
            else:
                self.cal_conf(freq_set, H1, sup_rata, strong_rules_list)
    return strong_rules_list
```

4. 分析挖掘结果

由于电脑配置问题，跑完全部数据花费的时间过于冗长。于是决定取前 100000 个数据进行实验。

```
return data_all.head(100000)
```

我们将得到的频繁项集放到了./results/频繁项集.json 文件中，按照支持度由大到小排列，形式如下图所示：


```
{"set": [["Agency", "OP"]], "sup": 1.0}
{"set": [["Priority", 2.0]], "sup": 0.80188}
{"set": [["Priority", 2.0], ["Agency", "OP"]], "sup": 0.80188}
{"set": [["Area Id", 1.0]], "sup": 0.36544}
{"set": [["Area Id", 1.0], ["Agency", "OP"]], "sup": 0.36544}
{"set": [["Area Id", 3.0]], "sup": 0.32649}
{"set": [["Agency", "OP"], ["Area Id", 3.0]], "sup": 0.32649}
{"set": [["Area Id", 2.0]], "sup": 0.30807}
{"set": [["Area Id", 2.0], ["Agency", "OP"]], "sup": 0.30807}
{"set": [["Priority", 2.0], ["Area Id", 1.0]], "sup": 0.29778}
{"set": [["Priority", 2.0], ["Area Id", 1.0], ["Agency", "OP"]], "sup": 0.29778}
{"set": [["Priority", 2.0], ["Area Id", 3.0]], "sup": 0.2557}
{"set": [["Priority", 2.0], ["Agency", "OP"], ["Area Id", 3.0]], "sup": 0.2557}
{"set": [["Priority", 2.0], ["Area Id", 2.0]], "sup": 0.2484}
{"set": [["Priority", 2.0], ["Area Id", 2.0], ["Agency", "OP"]], "sup": 0.2484}
{"set": [["Priority", 1.0]], "sup": 0.19811}
{"set": [["Priority", 1.0], ["Agency", "OP"]], "sup": 0.19811}
```


将得到的关联规则以及评价结果放到了./results/规则.json 文件中，按照置信度由大到小排列，形式如下图所示：


```
{"X_set": [["Area Id", 3.0]], "Y_set": [["Agency", "OP"]], "sup": 0.32649, "conf": 1.0, "lift": 1.0, "jaccard": 0.32649}
{"X_set": [["Area Id", 2.0]], "Y_set": [["Agency", "OP"]], "sup": 0.30807, "conf": 1.0, "lift": 1.0, "jaccard": 0.30807}
{"X_set": [["Priority", 2.0]], "Y_set": [["Agency", "OP"]], "sup": 0.80188, "conf": 1.0, "lift": 1.0, "jaccard": 0.80188}
{"X_set": [["Area Id", 1.0]], "Y_set": [["Agency", "OP"]], "sup": 0.36544, "conf": 1.0, "lift": 1.0, "jaccard": 0.36544}
{"X_set": [["Priority", 1.0]], "Y_set": [["Agency", "OP"]], "sup": 0.19811, "conf": 1.0, "lift": 1.0, "jaccard": 0.19811}
{"X_set": [["Area Id", 1.0]], "Y_set": [["Priority", 2.0]], "sup": 0.29778, "conf": 0.8148533274956217, "lift": 1.0161786395665457, "jaccard": 0.3424569312510062}
{"X_set": [["Area Id", 1.0]], "Y_set": [["Priority", 2.0], ["Agency", "OP"]], "sup": 0.29778, "conf": 0.8148533274956217, "lift": 1.0161786395665457, "jaccard": 0.3424569312510062}
{"X_set": [["Area Id", 2.0]], "Y_set": [["Priority", 2.0]], "sup": 0.2484, "conf": 0.8063102541630149, "lift": 1.0055248343430625, "jaccard": 0.2883175671754396}
{"X_set": [["Area Id", 2.0]], "Y_set": [["Priority", 2.0], ["Agency", "OP"]], "sup": 0.2484, "conf": 0.8063102541630149, "lift": 1.0055248343430625, "jaccard": 0.2883175671754396}
{"X_set": [["Agency", "OP"]], "Y_set": [["Priority", 2.0]], "sup": 0.80188, "conf": 0.80188, "lift": 1.0, "jaccard": 0.80188}
{"X_set": [["Area Id", 3.0]], "Y_set": [["Priority", 2.0]], "sup": 0.2557, "conf": 0.7831786578455695, "lift": 0.9766781287045062, "jaccard": 0.2930088120366232}
{"X_set": [["Area Id", 3.0]], "Y_set": [["Priority", 2.0], ["Agency", "OP"]], "sup": 0.2557, "conf": 0.7831786578455695, "lift": 0.9766781287045062, "jaccard": 0.2930088120366232}
```


由于所有的 Agency 属性的值都是 OP，所以对其分析没有实际意义，我们跳过包含 Agency 属性的频繁项集与规则进行分析。

我们可以由频繁项集.json 得知，Area Id 为 1.0 时支持度最高，

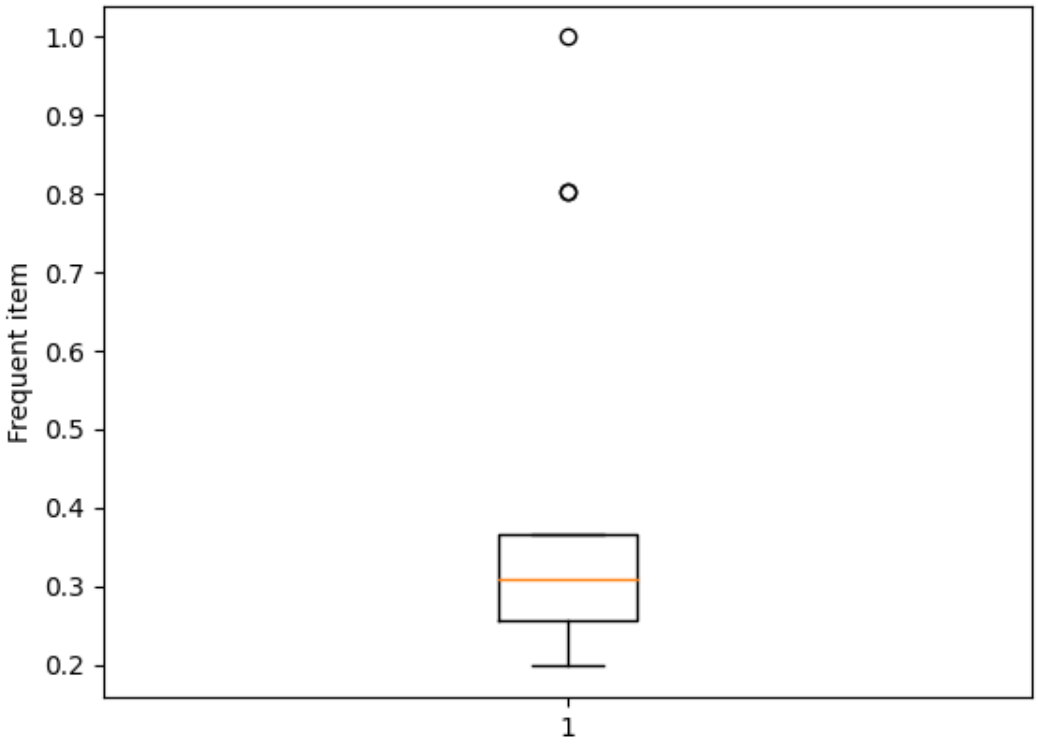
也就是说在该地区的犯罪事实出现最多。而且 Area Id 和 Priority 的关联度较高。

我们可以由规则.json 得知，["Area Id", 1.0]与["Priority", 2.0]的置信度较高，这说明犯罪的严重性与所在地有着较强联系。

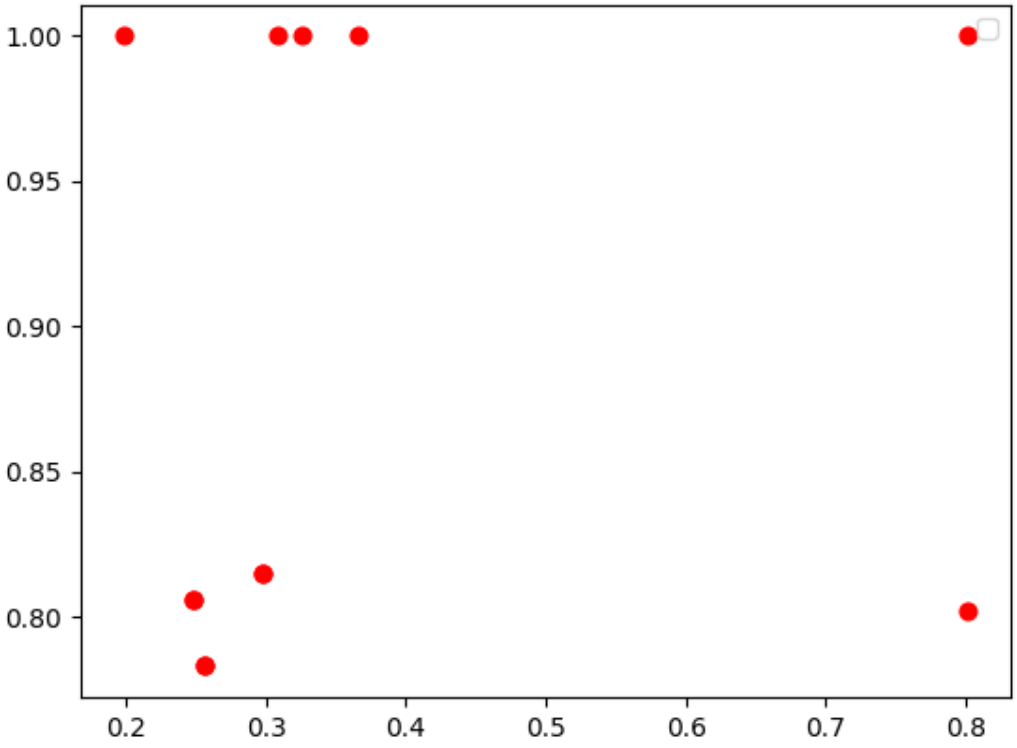
5. 可视化

分别使用盒图与散点图对频繁项集与规则进行可视化。

对频繁项集使用盒图可视化可得：



对规则的指出度与置信度使用散点图可视化可得：


1.00
0.95
0.90
0.85
0.80
0.2
0.3
0.4
0.5
0.6
0.7
0.8