姓名：张国雷
学号：3220201004
项目地址：https://github.com/zhangguolei/TheSecondHomework.git

# 一、数据集选择与挖掘目的

## 1.1 数据集

选择的数据集是奥克兰的犯罪数据集，Oakland Crime Statistics 2011 to 2016。该数据集一共有六个文件，分别保存了 2011 年到 2016 年间的犯罪数据。某些数据项缺失了，这里直接将缺失数据项的数据删除，不作为挖掘对象。数据集的属性如下所示：

Agency、Create Time、Location、Area Id、Beat、Priority、Incident Type Id、Incident Type Description、Event Number、Closed Time。

## 1.2 挖掘目的

虽然数据集的属性比较多，但是只需要重点关注若干个较为关键的属性，有 Agency（处理犯罪的机构）、Location（发生犯罪的地点）、Area Id（地区 ID）、Beat（打击数）、Incident Type Id（事件类型的 ID）、Priority（犯罪的严重等级）。

挖掘的目的就是要找到某些犯罪的特征，例如某个严重等级的犯罪在哪个地区出现的次数较高，或者哪个地区是犯罪高发区等等。

# 二、挖掘频繁模式

## 2. 1 挖掘算法介绍

使用的算法名称是 Apriori。Apriori 算法是第一个关联规则挖掘算法，也是最经典的算法。它利用逐层搜索的迭代方法找出数据库中项集的关系，以形成规则，其过程由连接（类矩阵运算）与剪枝（去掉那些没必要的中间结果）组成。该算法中项集的概念即为项的集合。包含 K 个项的集合为 k 项集。项集出现的频率是包含项集的事务数，称为项集的频率。如果某项集满足最小支持度，则称它为频繁项集[1]。

本实验中，采用 Apriori 算法创建频繁项集，支持度为 10%（需要超过全部数据集大小的 10%才可以被称为频繁项集），置信度为 50%（频繁项集内部的数据要达到这个概率）。

本实验中，Apriori 算法的核心代码如下所示。

```
def CountApriori(self, Dataset):
    C1 = self.C1Gen(Dataset)
    Dataset = [set(data) for data in Dataset]
    F1, SupRate = self.CkSupportFilter(Dataset, C1)
    F = [F1]
    k = 2
    while len(F[k-2]) > 0:
        Ck = self.ApGen(F[k-2], k)
        Fk, SupK = self.CkSupportFilter(Dataset, Ck)
        SupRate.update(SupK)
        F.append(Fk)
        k += 1
    return F, SupRate
```

生成单个元素的代码如下：

```
def C1Gen(self, Dataset):
    C1 = []
    progress = ProgressBar()
```

```
        for data in progress(Dataset):
            for i in data:
                if [i] not in C1:
                    C1.append([i])
        return [frozenset(i) for i in C1]
```

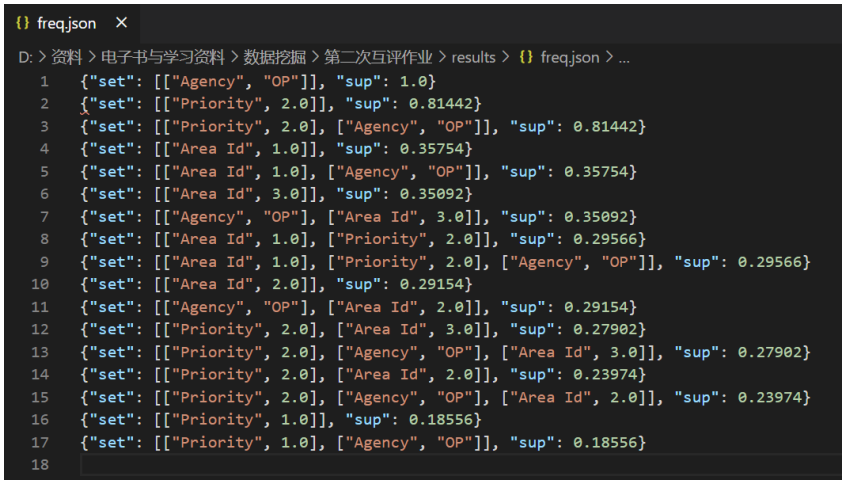
下面将低与设置的支持度的元素删除，代码如下：

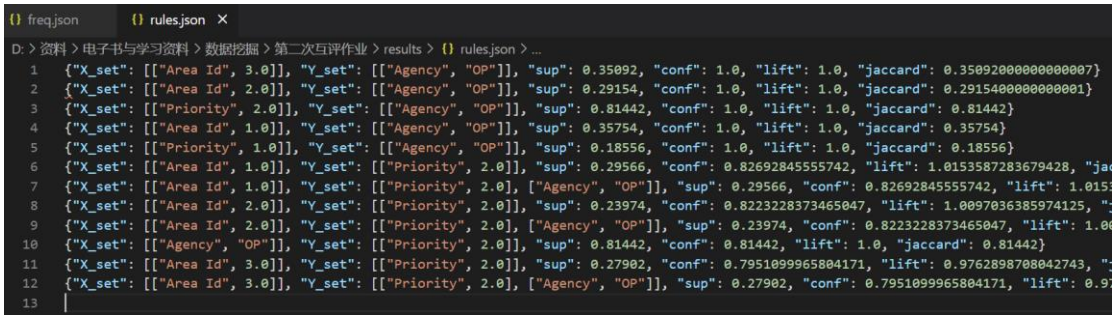
```
    def CkSupportFilter(self, Dataset, Ck):
        CompuCk = dict()
        for data in Dataset:
            for cand in Ck:
                if cand.issubset(data):
                    if cand not in CompuCk:
                        CompuCk[cand] = 1
                    else:
                        CompuCk[cand] += 1

        num_items = float(len(Dataset))
        reLi = []
        SupRate = dict()

        for key in CompuCk:
            support    = CompuCk[key] / num_items
            if support >= self.minSupport:
                reLi.insert(0, key)
            SupRate[key] = support
        return reLi, SupRate
```

将得出的 k-1 长度的项集进行合并，得到 k 长度的项集，然后筛选出符合条件的项集，组成新的集合，代码如下：

```
    def ApGen(self, Fk, k):
        reLi = []
        Flength = len(Fk)

        for i in range(Flength):
            for j in range(i+1, Flength):
                F1 = list(Fk[i])[:k-2]
                F2 = list(Fk[j])[:k-2]
                F1.sort()
                F2.sort()
                if F1 == F2:
                    reLi.append(Fk[i] | Fk[j])
        return reLi
```

得出然后，将 ApGen 函数生成的频繁项集导出到本地文件，放入到文件 freq.json 和 rules.json 中，如下所示。




## 三、生成关联规则

依据上述得出的频繁项集，导出关联规则，并且使用 Lift 和 Jaccard 指标进行评价。Lift 和 Jaccard 指标的评价公式如下所示。

Lift:

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

Jaccard:

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

下面是使用 Lift 和 Jaccard 指标进行评价的代码：

```
def CountConfigure(self, freSet, H, SupRate, StroRule):
    tempH = []
    for item in H:
        sup = SupRate[freSet]
        conf = sup / SupRate[freSet - item]
        lift = conf / SupRate[item]
        jaccard = sup / (SupRate[freSet - item] + SupRate[item] - sup)
        if conf >= self.minConfigure:
            StroRule.append((freSet-item, item, sup, conf, lift, jaccard))
            tempH.append(item)
    return tempH
```

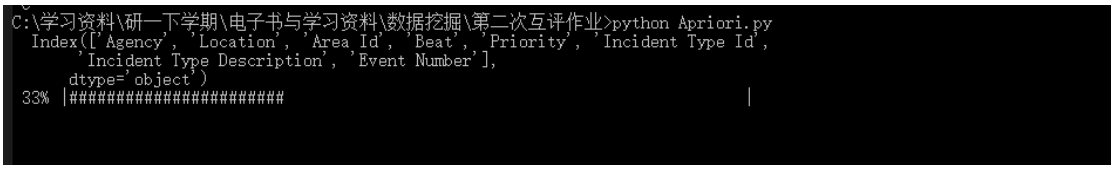
接着需要保留达到要求的项集，代码如下：

```
    def GenRule(self, F, SupRate):
        StroRule = []
```

```
        for i in range(1, len(F)):
            for freSet in F[i]:
                H1 = [frozenset([item]) for item in freSet]
                if i > 1:
                    self.RuleItem(freSet, H1, SupRate, StroRule)
                else:
                    self.CountConfigure(freSet, H1, SupRate, StroRule)
        return StroRule
```

## 四、挖掘的过程和结果

取前 10000 条数据进行实验。运行过程如下图：





如前所述，实验的结果保存在了 results 文件夹中的 freq.json 和 rules.json 中。

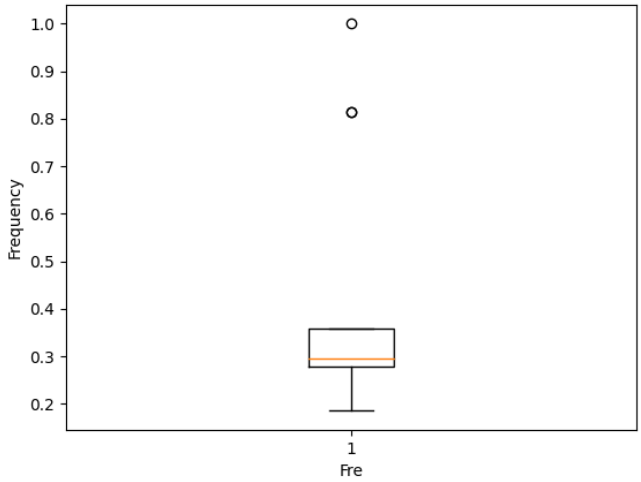
首先分析导出的 freq.json 数据可得，Area Id=1 的时候，犯罪的频率最高。其次，Area Id=3 的时候犯罪率位居第二。同时，由于数据项{"set": [["Area Id", 1.0], ["Priority", 2.0]], "sup": 0.29566}的支持度很高，表示 Area Id=1 的地区很容易发生 Priority=2.0 的犯罪，二者的关联度比较高。

然后分析导出的 rules.json 数据，可以得出 Agency=”OP”经常处理一些 Area Id=1.0，2.0，3.0 的犯罪事件。其次，Area Id=1.0，2.0，3.0 的地方经常发生 Priority=2.0 的犯罪。

## 五、挖掘结果的可视化呈现

### 5.1 可视化频繁项集

使用盒图对频繁项集进行可视化，结果如下图所示。



接着，对支持度和置信度使用散点图绘制，得到的结果如下图所示。