数据挖掘大作业二：关联规则挖掘

姓名：白雪峰　　学号：3120180977

1 数据说明

1.1 数据集

选择数据集：OaklandCrimeStatistics2011to2016

1.2 数据集描述

数据集：OaklandCrimeStatistics2011to2016
包含 records-for-2011.csv 到 records-for-2016.csv 共六个文件
属性列表：
Agency，CreateTime，Location，AreaId，Beat，Priority，IncidentTypeId，IncidentType，Description，EventNumber，ClosedTime

2 问题描述

对 OaklandCrimeStatistics2011to2016 数据集进行关联规则挖掘。共包含以下五个子任务：

- 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 及其它指标,要求至少 2 种；
- 对挖掘结果进行可视化展示。

3 关联规则挖掘过程

3.1 处理数据集

第一步要将数据集处理成适合关联规则挖掘的格式，考虑到数据的完整性以及挖掘过程的复杂度和运算速度，这里选择了：

Agency
Location

Area Id
Beat
Priority
Incident Type Id
Incident Type Description
Event Number

对这些属性进行关联规则挖掘属于多维关联挖掘的问题，而单维规则分析问题仅仅针对单个属性，因此我们需要将属于不同属性的值转化为可以产生频繁项集的形式。过程如下：

首先将 2011 年值 2016 年的 6 个 csv 文件分别读取并整合为同一个 DataFrame 类：

```
def dataread(self):
    for year in range(2011, 2017):
        print("year:", year)
        dataframe = pd.read_csv(data_file_path + "/records-for-{}.csv".format(str(year)))
        columnsList = list(dataframe)
        if "Location 1" in columnsList:
            dataframe.rename(columns={"Location 1": "Location"}, inplace=True)
        elif "Location " in columnsList:
            dataframe.rename(columns={"Location ": "Location"}, inplace=True)
        order = ["Agency", "Create Time", "Location", "Area Id", "Beat", "Priority", "Incident Type Id", "Incident Type Description", "Event Number", "Closed Time"]
        newdf = dataframe[order]
        resdf = pd.DataFrame(newdf, columns=["Agency", "Location", "Area Id", "Beat", "Priority", "Incident Type Id", "Incident Type Description", "Event Number"])
        reslist = resdf.values.tolist()
        reslist.insert(0, ["Agency", "Location", "Area Id", "Beat", "Priority", "Incident Type Id", "Incident Type Description", "Event Number"])

        return reslist
```

然后将每一个属性名与属性值的组合表示为一个元组的形式：（属性名，属性值）。作为一个单项。并使用 python 中的 frozenset 类型表示项集。与 set 类型不同的是，frozenset 类型一经创建便不可修改，因此可以将可迭代对象转化为字典的键进行处理。

```
dataset = []
feature_names = rows[0]
for data_line in rows[1:]:
    data_set = []
    for i, value in enumerate(data_line):
        if not value:
            data_set.append((feature_names[i], 'NA'))
        else:
            data_set.append((feature_names[i], value))
    dataset.append(data_set)
```

对于数据进行读取和处理的代码分别位于 data_read.py 和 data.py 文件中。

3.2 产生频繁项集

使用 Apriori 算法在经过预处理的数据集上构建频繁项集。频繁项集表示在数据集中共现频率高的属性项的集合。关联规则的强度主要可以使用两个指标来衡量：支持度和置信度。支持度表示一个项集或者规则在所有事物中出现的频率，确定规则可以用于给定数据集的频繁程度，X->Y 的支持度指的就是 X 和 Y 共同出现的概率。X->Y 的置信度表示 Y 在包含 X 的事务中出现的频繁程度，也就是在出现了 X 的项中，Y 出现的概率有多大。

首先为这两个超参数设定阈值，设定最小支持度为 0.01（将支持度设得偏小是因为数据集中有几个属性的取值较为分散，如果支持度太高则会忽略掉这些属性），将最小置信度设为 0.5。

```
def __init__(self, min_support = 0.5, min_confidence = 0.9):
    self.min_support = min_support          # 最小支持度
    self.min_confidence = min_confidence    #  最小置信度
```

Apriori 算法使用频繁项集的先验知识，使用一种称作逐层搜索的迭代方法，k 项集用于探索(k+1)项集（如果事件 A 中包含 k 个元素，那么称这个事件 A 为 k 项集，并且事件 A 满足最小支持度阈值的事件称为频繁 k 项集）。首先，通过扫描事务（交易）记录，找出所有的频繁 1 项集，该集合记做 L1，然后利用 L1 找频繁 2 项集的集合 L2，L2 找 L3，如此下去，直到不能再找到任何频繁 k 项集。

Apriori 算法流程如下：

- 扫描一次数据库 D；计算出各个 1 项集的支持度，得到频繁 1 项集的集合。
- 从 2 项集开始循环，进行由频繁 k-1 项集生成频繁频繁 k 项集。
  - 连接步：将 2 个只有一个项不同的属于的频集做一个（k-2）JOIN 运算得到。
  - 剪枝步：因为是超集，所以可能有些元素不是频繁的。舍弃掉子集不是频繁项集即不在频繁 k-1 项集中的项集
  - 扫描数据库，计算 2.3 步中过滤后的 k 项集的支持度，舍弃掉支持度小于阈值的项集，生成频繁 k 项集。
- 当生成的频繁 k 项集中只有一个项集时循环结束

Apriori 算法的实现位于“associasion.py”，程序代码如下：

- Apriori 主函数：

```
def apriori(self, dataset):
    """
    Apriori算法实现
    :param dataset: 数据集，类型为一个list，list中每个元素是一个dict，key为属性名，value为对应属性的取值
    :return: 生成频繁项集
    """
    C1 = self.create_C1(dataset)
    dataset = [set(data) for data in dataset]
    L1, support_data = self.scan_D(dataset, C1)
    L = [L1]
    k = 2
    while len(L[k-2]) > 0:
        Ck = self.apriori_gen(L[k-2], k)
        Lk, support_k = self.scan_D(dataset, Ck)
        print(Lk)
        support_data.update(support_k)
        L.append(Lk)
        k += 1
    return L, support_data
```

- create_C1()函数用于生成初始的单个项项集集合：

```
def create_C1(self, dataset):
    """
    扫描dataset，构建全部可能的单元素候选项集合(list)
    每个单元素候选项：（属性名，属性取值）
    """
    C1 = []
    for data in tqdm(dataset):
        for item in data:
            if [item] not in C1:
                C1.append([item])
    return [frozenset(item) for item in C1]
```

- Scan_D()函数用于扫描项集集合，并过滤掉小于最小支持度的项集：

```
def scan_D(self, dataset, Ck):
    """
    过滤函数
    根据待选项集Ck的情况，判断数据集D中Ck元素的出现频率
    过滤掉低于最小支持度的项集
    """
    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 = []
    support_data = dict()
    # 过滤非频繁项集
    for key in Ck_count:
        support = Ck_count[key] / num_items
        if support >= self.min_support:
            return_list.insert(0, key)
        support_data[key] = support
    return return_list, support_data
```

- Apriori_gen()函数用于非重复地合并两个项集：

```
def apriori_gen(self, Lk, k):
    """
    当待选项集不是单个元素时， 如k>=2的情况下，合并元素时容易出现重复
    因此针对包含k个元素的频繁项集，对比每个频繁项集第k-2位是否一致
    """
    return_list = []
    len_Lk = len(Lk)

    for i in range(len_Lk):
        for j in range(i+1, len_Lk):
            # 第k-2个项相同时，将两个集合合并
            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
```

3.3 导出关联规则

使用上面基于 Apriori 算法产生的频繁项集，进行强关联规则的挖掘。过程如下：

- 根据选定的频繁项集，找到它所有的非空子集。然后建立规则列表。
- 强关联规则需要满足最小支持度和最小置信度，对每一条规则计算指标：
  - 支持度：

$$Sup(X) = \frac{Sum(X)}{N}$$

  - 置信度：

$$Conf\left(X \Rightarrow Y\right) = \frac{Sup(X \cup Y)}{Sup(X)}$$

  - 对每一条规则还需要计算提升度指标：

$$Lift\left(X \Rightarrow Y\right) = \frac{Sup(X \cup Y)}{Sup(X) \times Sup(Y)} = \frac{Conf(X \cup Y)}{Sup(Y)}$$

    表示“包含 A 的事务中同时包含 B 的事务的比例”与“包含 B 的事务的比例”的比值。用来判断规则 X⇒Y 中的 X 和 Y 是否独立，如果独立，那么这个规则是无效的。提升度反映了关联规则中的 A 与 B 的相关性，提升度 >1 且越高表明正相关性越高，提升度 < 1 且越低表明负相关性越高，提升度 = 1 表明没有相关性。但是在具体的应用之中，我们认为提升度 > 3 才算作值得认可的关联。
- 根据以上指标，找到所有可能的关联规则。

产生强关联规则的算法同样位于“associasion.py”的 Associasion 类中，程序代码如下：

- Rules_from_conseq()函数，用于递归地产生规则右部的结果项集：

```
def rules_from_conseq(self, freq_set, H, support_data, big_rules_list):
    """
    H->出现在规则右部的元素列表
    """
    m = len(H[0])
    if len(freq_set) > (m+1):
        Hmp1 = self.apriori_gen(H, m+1)
        Hmp1 = self.cal_conf(freq_set, Hmp1, support_data, big_rules_list)
        if len(Hmp1) > 1:
            self.rules_from_conseq(freq_set, Hmp1, support_data, big_rules_list)
```

- Generate_rules()函数，用于产生强关联规则：

```python
def generate_rules(self, L, support_data):
    """
    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 L: 频繁项集
    :param support_data: 频繁项集对应的支持度
    :return: 强关联规则列表
    """
    big_rules_list = []
    for i in range(1, len(L)):
        for freq_set in L[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_conseq(freq_set, H1, support_data, big_rules_list)
            else:
                self.cal_conf(freq_set, H1, support_data, big_rules_list)
    return big_rules_list
```

- Cal_conf()函数，用于评价生成的规则，并计算支持度、置信度、lift 指标：

```python
def cal_conf(self, freq_set, H, support_data, big_rules_list):
    """
    评估生成的规则
    """
    prunedH = []
    for conseq in H:
        sup = support_data[freq_set]
        conf = sup / support_data[freq_set - conseq]
        lift = conf / support_data[freq_set - conseq]
        if conf >= self.min_confidence:
            big_rules_list.append((freq_set-conseq, conseq, sup, conf, lift))
            prunedH.append(conseq)
    return prunedH
```

3.4 挖掘结果及分析

挖掘出的频繁项集按照每一项一行的格式存储在“./results/freq_set.json”文件下，并且将频繁项集按照支持度从大到小进行排列，以下图为例说明输出结果格式：


```
{
    "set": [["Incident Type Description", "ALARM-RINGER"],
            ["Agency", "OP"],
            ["Priority", 2.0]],
    "sup": 0.4044044044044044
}
```

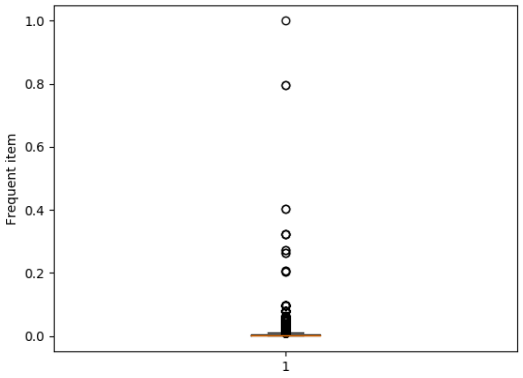

挖掘出的符合条件的关联规则

```
{
    "X_set": [["Beat", "18Y"]],
    "Y_set": [["Agency", "OP"], ["Area Id", 2.0]],
    "sup": 0.011011011011011011,
    "conf": 1.0,
    "lift": 90.81818181818181
}
```


通过对最终挖掘得到的关联规则进行分析我们可以得到，“Beat”属性与“Agency”属性的属性关联度极高，并且“Area Id-> Incident Type Id”规则的置信度极高，这说明大部分犯罪事件的类型与发生的地点有着很大的联系，犯罪事件有很强的地域特征。

4 结果可视化

对频繁项集结果做可视化



对关联规则挖掘结果做可视化