> 3220201026
>
> 郑慧娴
>
> 计算机学院

数据挖掘-互评作业2

1.数据分析要求

- 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；

2.挖掘过程

2.1 数据集

Oakland Crime Statistics 2011 to 2016

2.2 处理数据集

该数据集中包括6个子数据集，分别是从2011-2016年的犯罪记录。读入数据集，并查看每个子数据集的属性，代码和结果如下：

```
data2011 = pd.read_csv("C:/Users/hespe/Desktop/课件/数据挖掘/archive/records-for-2011.csv", encoding="utf-8")
data2012 = pd.read_csv("C:/Users/hespe/Desktop/课件/数据挖掘/archive/records-for-2012.csv", encoding="utf-8")
data2013 = pd.read_csv("C:/Users/hespe/Desktop/课件/数据挖掘/archive/records-for-2013.csv", encoding="utf-8")
data2014 = pd.read_csv("C:/Users/hespe/Desktop/课件/数据挖掘/archive/records-for-2014.csv", encoding="utf-8")
data2015 = pd.read_csv("C:/Users/hespe/Desktop/课件/数据挖掘/archive/records-for-2015.csv", encoding="utf-8")
data2016 = pd.read_csv("C:/Users/hespe/Desktop/课件/数据挖掘/archive/records-for-2016.csv", encoding="utf-8")

print("2011数据集有以下属性", data2011.columns)
print("2012数据集有以下属性", data2012.columns)
print("2013数据集有以下属性", data2013.columns)
print("2014数据集有以下属性", data2014.columns)
print("2015数据集有以下属性", data2015.columns)
print("2016数据集有以下属性", data2016.columns)
```

```
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')
```

根据结果可以看出，6个子数据集的数据属性基本一样。可以进行分析和预处理的属性包括：'Agency'，'Location'，'Area Id'，'Beat'，'Incident Type Id'，'Incident Type Describe'，'Event Number' 。

其中2012年、2014年Location属性为'Location 1'，将其处理为'Location'；2013年Location属性为'Location '，将其处理为'Location'。'Incident Type Id'与'Incident Type Describe'是一一对应的，因此只分析'Incident Type Id'，'Event Number' 不具备重复性，不符合频繁模式挖掘的要求，不对其分析。

提取每个子数据集中上述属性的数据，将6个子集数据集成为综合数据集，代码和结果如下：

```
data2012.rename(columns={"Location 1": "Location"}, inplace=True)
data2013.rename(columns={"Location ": "Location"}, inplace=True)
data2014.rename(columns={"Location 1": "Location"}, inplace=True)

data2011_temp = data2011[
    ["Agency", "Location", "Area Id", "Beat", "Priority", "Incident Type Id",
"Incident Type Description",
     "Event Number"]]
data2012_temp = data2012[
    ["Agency", "Location", "Area Id", "Beat", "Priority", "Incident Type Id",
"Incident Type Description",
     "Event Number"]]
data2013_temp = data2013[
    ["Agency", "Location", "Area Id", "Beat", "Priority", "Incident Type Id",
"Incident Type Description",
     "Event Number"]]
data2014_temp = data2014[
    ["Agency", "Location", "Area Id", "Beat", "Priority", "Incident Type Id",
"Incident Type Description",
     "Event Number"]]
data2015_temp = data2015[
    ["Agency", "Location", "Area Id", "Beat", "Priority", "Incident Type Id",
"Incident Type Description",
     "Event Number"]]
data2016_temp = data2016[
```

```
    ["Agency", "Location", "Area Id", "Beat", "Priority", "Incident Type Id",
"Incident Type Description",
     "Event Number"]]

data_all = pd.concat([data2011_temp, data2012_temp, data2013_temp,
data2014_temp, data2015_temp, data2016_temp],
                    axis=0)
print("综合数据集有以下属性", data_all.columns)
data_all = data_all.dropna(how='any')
```

```
综合数据集有以下属性 Index(['Agency', 'Location', 'Area Id', 'Beat', 'Priority', 'Incident Type Id',
       'Incident Type Description', 'Event Number'],
      dtype='object')
```

综合数据集共计1046388条，对于有缺失数据的记录，采用上次互评作业的方法，舍弃有缺失值的行，最终得到的数据为859898条。

2.3频繁模式挖掘

在本实验中，使用Apriori算法来构建频繁项集。

2.3.1 Apriori算法

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。算法主要流程如下：

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

在本实验中，设置支持度的阈值为10%，置信度的阈值为50%。

```
min_sup = 0.1  # 设置最小支持度
min_conf = 0.5  # 设置最小置信度
```

2.3.2 算法实现过程

生成单元数候选项集：

```
def C1_generation(self, dataset):  # 生成单元数候选项集
    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]
```

过滤支持度低于阈值的项集：

```
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
```

当候选项元素大于2时，合并时检测是否子项集满足频繁：

```
def apriori_gen(self, Fk, k):  # 当候选项元素大于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
```

Apriori算法：

```
def apriori(self, dataset):  # Apriori算法
    C1 = self.C1_generation(dataset)  # 生成单元数候选项集
    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)  # 当候选项元素大于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
```

将得到的频繁项集输出到结果文件'sup_rata.json'：

```
# 获取频繁项集
freq_set, sup_rata = association.apriori(dataset)
sup_rata_out = sorted(sup_rata.items(), key=lambda d: d[1], reverse=True)
print("sup_rata ", sup_rata)
```

```
# 将频繁项集输出到结果文件
freq_set_file = open(os.path.join(out_path, 'sup_rata.json'), 'w')
for (key, value) in sup_rata_out:
    result_dict = {'set': None, 'sup': None}
    set_result = list(key)
    sup_result = value
    if sup_result < min_sup:
        continue
    result_dict['set'] = set_result
    result_dict['sup'] = sup_result
    json_str = json.dumps(result_dict, ensure_ascii=False)
    freq_set_file.write(json_str + '\n')
freq_set_file.close()
```

2.4导出关联规则

2.4.1算法实现过程

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。

```
def generate_rules(self, F, sup_rata):
    """
    :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

def rules_from_reasoned_item(self, freq_set, H, sup_rata, strong_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, sup_rata, strong_rules_list)
        if len(Hmp1) > 1:
```

```
            self.rules_from_reasoned_item(freq_set, Hmp1, sup_rata,
strong_rules_list)
```

```
# 获取强关联规则列表
strong_rules_list = association.generate_rules(freq_set, sup_rata)
strong_rules_list = sorted(strong_rules_list, key=lambda x: x[3], reverse=True)
print("strong_rules_list ", strong_rules_list)
```

```
# 将关联规则输出到结果文件
rules_file = open(os.path.join(out_path, 'strong_rules_list.json'), 'w')
for result in strong_rules_list:
    result_dict = {'X_set': None, 'Y_set': None, 'sup': None, 'conf': None,
'lift': None, 'jaccard': None}
    X_set, Y_set, sup, conf, lift, jaccard = result
    result_dict['X_set'] = list(X_set)
    result_dict['Y_set'] = list(Y_set)
    result_dict['sup'] = sup
    result_dict['conf'] = conf
    result_dict['lift'] = lift
    result_dict['jaccard'] = jaccard
    json_str = json.dumps(result_dict, ensure_ascii=False)
    rules_file.write(json_str + '\n')
rules_file.close()
```

2.4.2评价指标

2.4.2.1 Lift系数

相关性系数的英文名是Lift，这就是一个单词，而不是缩写。通俗解释：提升度反映了“物品集X XX的出现”对物品集Y YY的出现概率发生了多大的变化。

$$\mathrm{lift}(X \Rightarrow Y) = \frac{\mathrm{supp}(X \cup Y)}{\mathrm{supp}(X) \times \mathrm{supp}(Y)}$$

$$\mathrm{lift}(X \Rightarrow Y) \begin{cases} > 1, & \text{正相关} \\ = 1, & \text{独立} \\ < 1, & \text{负相关} \end{cases}$$

2.4.2.2 卡方系数

卡方系数是与卡方分布有关的一个指标。公式中的O_i表示数据的实际值，E_i表示期望值。

$$\chi^2 = \sum_{i=1}^{n} \frac{(O_i - E_i)^2}{E_i}$$

2.4.2.3代码实现

```
def cal_conf(self, freq_set, H, sup_rata, strong_rules_list):  # 评价指标
    prunedH = []
    for reasoned_item in H:
        sup = sup_rata[freq_set]  # 支持度
        conf = sup / sup_rata[freq_set - reasoned_item]  # 置信度
        lift = conf / sup_rata[reasoned_item]  # lift
        jaccard = sup / (sup_rata[freq_set - reasoned_item] +
sup_rata[reasoned_item] - sup)  # JACCARD
        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
```

3.挖掘结果及可视化

3.1挖掘结果及分析

由于数据量太大，本实验只采用前100000条数据进行频繁模式和关联规则挖掘。

得到的频繁模式根据支持度从大到小，保存下'./results/sup_rata.json'中：

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

根据频繁项集文件'sup_rata.json'，‘Agency’属性的值都是‘OP’，因此没有分析的意义，跳过。

‘Area Id’=1.0时支持度最高，说明该地区犯罪记录最多。且‘Area Id’和‘Priority’的关联度较高。

导出关联规则根据置信度从大到小，保存在‘./results/strong_rules_list.json’中：

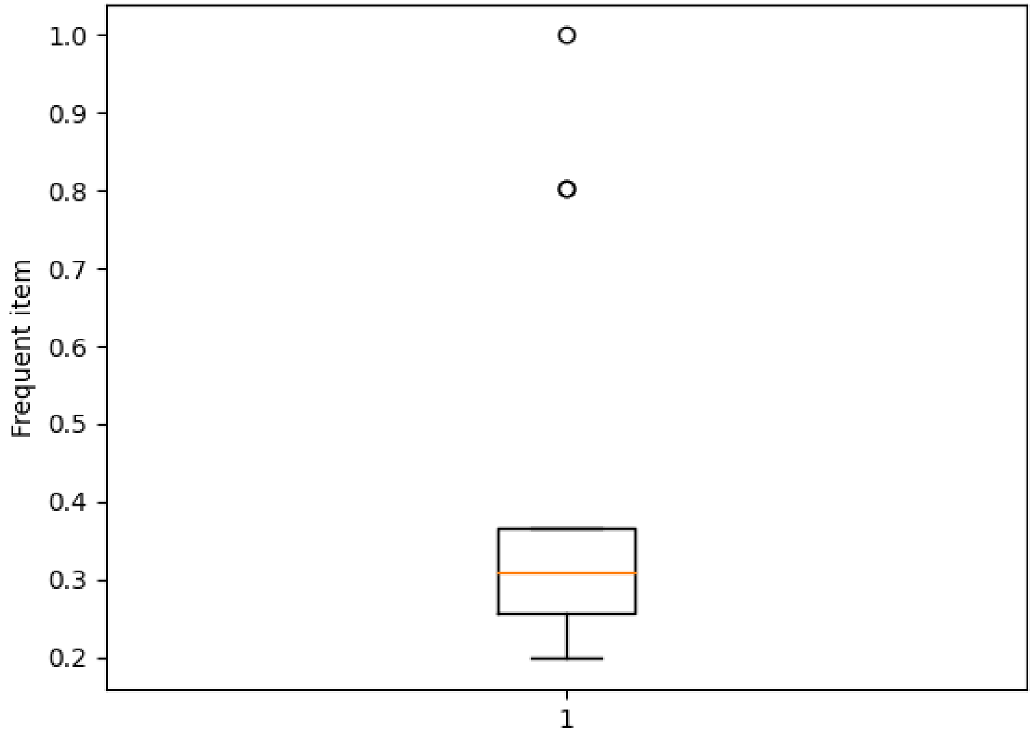
{"X_set": [["Area Id", 3.0]], "Y_set": [["Agency", "OP"]], "sup": 0.32649, "conf": 1.0, "lift": 1.0, "jaccard": 0.32649}
{"X_set": [["Area Id", 2.0]], "Y_set": [["Agency", "OP"]], "sup": 0.30807, "conf": 1.0, "lift": 1.0, "jaccard": 0.30807}
{"X_set": [["Priority", 2.0]], "Y_set": [["Agency", "OP"]], "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}

根据关联规则文件‘strong_rules_list.json’，["Area Id",1.0]与["Priority",2.0]的置信度较高，说明犯罪严重性与犯罪所在地区有较强联系。

3.2可视化

```
class Visualization():

    with open("./results/sup_rata.json") as f1:
        freq = [json.loads(each) for each in f1.readlines()]

    with open("./results/strong_rules_list.json") as f2:
        rules = [json.loads(each) for each in f2.readlines()]

    freq_sup = [each["sup"] for each in freq]
    plt.boxplot(freq_sup)
    plt.ylabel("Frequent item")
    plt.show()

    rules_sup = [each["sup"] for each in rules]
    rules_conf = [each["conf"] for each in rules]

    plt.scatter(rules_sup, rules_conf, marker='o', color='red', s=40)
    plt.xlabel = 'Sup'
    plt.ylabel = 'Conf'
    plt.legend(loc='best')
    plt.show()
```

采用盒图对频繁项集可视化:



采用散点图对关联规则可视化：