数据挖掘互评作业二: 频繁模式与关联规则挖掘

3220211037 刘秉杰

提交说明

1. 问题描述

本次作业中，将选择1个数据集进行频繁模式和关联规则挖掘。

2. 数据说明

- Consumer & Visitor Insights For Neighborhoods
- Wine Reviews
- Oakland Crime Statistics 2011 to 2016
- Chicago Building Violations
- Trending YouTube Video Statistics
- Melbourne Airbnb Open Data
- MLB Pitch Data 2015-2018

与第一次互评作业的数据集范围是相同的，在选择的时候可以选择之前预处理的数据集，也可以重新选择一个。

3. 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。

4. 提交的内容

- 对数据集进行处理的代码
- 关联规则挖掘的代码
- 挖掘过程的报告：展示挖掘的过程、结果和你的分析
- 所选择的数据集在README中说明，数据文件不要上传到Github中

乐学平台提交注意事项：

- 仓库地址：**记得加上**
- 报告：附件，word，pdf，html格式都可以

作业说明

数据集

Oakland Crime Statistics 2011 to 2016

使用说明

Python版本：Python 2.7.17

操作系统：Ubuntu

命令行中输入以下命令来运行：

```
mkdir figs
rm ./figs/*
mkdir fpar_results
rm ./fpar_results/*
python2 fpar.py
```

代码仓库：https://github.com/Benjamin-Lau/BITCS_DM2022

代码

```
# coding:utf-8
import matplotlib
matplotlib.use('Agg')

import numpy as np
import scipy as sp
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import json
import math
import re
import sys
import csv
import random
import os
from tqdm import tqdm

class Association(object):
    '''
    Find all frequent items,
    and generate strong association rules from frequent itemset
    '''
    def __init__(self, min_support = 0.1, min_confidence = 0.5):
        self.min_support = min_support          # minimum support
        self.min_confidence = min_confidence    # minimum confidence
    '''
    Apriori algorithm
    Input: dataset(list of dict)
    Return: L--frequent itemset,
            support_data--support corresponding to frequent itemset
    '''
    def apriori(self, dataset):
        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)
            support_data.update(support_k)
            L.append(Lk)
            k += 1
        return L, support_data

    '''
    Create set C1 which is the set of all possible candidate item
    Format of each item: (Attr, value)
    '''
    def create_C1(self, dataset):
        C1 = []
        for data in dataset:
            for item in data:
                if [item] not in C1:
                    C1.append([item])
        return [frozenset(item) for item in C1]

    '''
    Get the frequency of Ck elements in data set D
    according to the condition of option set CK to be selected,
    clear the item set with lower support than the minimum support
    '''
    def scan_D(self, 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()
        # clear non-frequent itemset
        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

    def apriori_gen(self, Lk, k):
        # for the frequent itemset with k items, compare the (k-2)th item
        return_list = []
        len_Lk = len(Lk)
        for i in range(len_Lk):
            for j in range(i+1, len_Lk):
                # if the (k-2)th items of two sets are the same, merge them
                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

    '''
    Generate rules
    Based on Apriori algorithm, first start with a frequent itemset,
    and then create a rule list,
    The right part of the rule contains only one element,
    and then these rules are tested.
    Next, merge all the remaining rule lists to create a new rule list,
    The right part of the rule contains two elements. This method is known as grading
method.
    Return: strong rules
    '''
    def generate_rules(self, L, support_data):
        strong_rules_list = []
        for i in range(1, len(L)):
            for freq_set in L[i]:
                H1 = [frozenset([item]) for item in freq_set]
                # Just convern the sets with 2 or more elements
                if i > 1:
                    self.rules_from_conseq(freq_set, H1, support_data,
strong_rules_list)
                else:
                    self.cal_conf(freq_set, H1, support_data, strong_rules_list)
        return strong_rules_list

    def rules_from_conseq(self, freq_set, H, support_data, strong_rules_list):
        # H are the items in the right part of the rules
        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, strong_rules_list)
            if len(Hmp1) > 1:
                self.rules_from_conseq(freq_set, Hmp1, support_data, strong_rules_list)

    def cal_conf(self, freq_set, H, support_data, strong_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 >= self.min_confidence:
                strong_rules_list.append((freq_set-conseq, conseq, sup, conf,
lift,Jaccard))
                prunedH.append(conseq)
        return prunedH

class oaklandCrimeStatistics():
    def __init__(self,data_file_path,result_path,feature_list=None):
        self.data_file_path = data_file_path
        self.feature_list = feature_list
        self.result_path = result_path

    def set_feature_list(self,feature_list):
        self.feature_list = feature_list

    def set_data_file_path(self,data_file_path):
        self.data_file_path = data_file_path

    def set_result_path(result_path):
        self.result_path = result_path

    def data_read(self):

        ocs2011 = pd.read_csv(self.data_file_path+'/records-for-2011.csv',
encoding='utf-8')
        ocs2012 = pd.read_csv(self.data_file_path+'/records-for-2012.csv',
encoding='utf-8')
        ocs2013 = pd.read_csv(self.data_file_path+'/records-for-2013.csv',
encoding='utf-8')
        ocs2014 = pd.read_csv(self.data_file_path+'/records-for-2014.csv',
encoding='utf-8')
        ocs2015 = pd.read_csv(self.data_file_path+'/records-for-2015.csv',
encoding='utf-8')
```

```
        ocs2016 = pd.read_csv(self.data_file_path+'/records-for-2016.csv',
encoding='utf-8')

        # 'Location' in 2013 is 'Location ', in 2012 & 2014 is 'Location 1'
        ocs2012.rename(columns={'Location 1': 'Location'}, inplace = True)
        ocs2013.rename(columns={'Location ': 'Location'}, inplace = True)
        ocs2014.rename(columns={'Location 1': 'Location'}, inplace = True)

        order=['Agency', 'Location', 'Area Id', 'Beat', 'Priority',
               'Incident Type Id', 'Incident Type Description', 'Event Number']
        _ocs2011 = ocs2011[order]
        _ocs2012 = ocs2012[order]
        _ocs2013 = ocs2013[order]
        _ocs2014 = ocs2014[order]
        _ocs2015 = ocs2015[order]
        _ocs2016 = ocs2016[order]

        ocs_all = pd.concat([_ocs2011, _ocs2012, _ocs2013, _ocs2014, _ocs2015,
_ocs2016],
                            axis=0)
        print('Merged Dataset: ')
        print(ocs_all.info())
        ocs_all = ocs_all.dropna(how='any')

        return ocs_all


    def mining(self,min_support = 0.1, min_confidence = 0.5,head_n=None):
        out_path = self.result_path
        association =
Association(min_support=min_support,min_confidence=min_confidence)
        ocs_all = self.data_read()
        rows = None
        if head_n is None:
          rows = ocs_all.values.tolist()
        else:
          rows = ocs_all.head(head_n).values.tolist()

        # save dict into json
        dataset = []
        feature_names = ['Agency', 'Location', 'Area Id', 'Beat', 'Priority',
                         'Incident Type Id', 'Incident Type Description', 'Event
Number']
        for data_line in rows:
            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)
        print('Mining start...')
        # get frequent itemset
        freq_set, sup_rata = association.apriori(dataset)
        sup_rata_out = sorted(sup_rata.items(), key=lambda d: d[1], reverse=True)
        # get strong rules
        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('Mining completed!')
        # save frequent itemset
        freq_set_file = open(os.path.join(out_path, 'freq.json'), 'w')
        for (key, value) in sup_rata_out:
            res_dict = {'set': None, 'sup': None}
            set_result = list(key)
            sup_result = value
            if sup_result < association.min_support:
                continue
            res_dict['set'] = set_result
            res_dict['sup'] = sup_result
            json_str = json.dumps(res_dict, ensure_ascii=False)
            freq_set_file.write(json_str + '\n')
            print(res_dict)
        freq_set_file.close()

        # save rules
        rules_file = open(os.path.join(out_path, 'rules.json'), 'w')
        for result in strong_rules_list:
            res_dict = {'X_set': None, 'Y_set': None, 'sup': None, 'conf': None,
'lift': None, 'Jaccard': None}
            X_set, Y_set, sup, conf, lift, Jaccard = result
            res_dict['X_set'] = list(X_set)
            res_dict['Y_set'] = list(Y_set)
            res_dict['sup'] = sup
            res_dict['conf'] = conf
            res_dict['lift'] = lift
            res_dict['Jaccard'] = Jaccard

            json_str = json.dumps(res_dict, ensure_ascii=False)
            rules_file.write(json_str + '\n')
            print(res_dict)
        rules_file.close()
        print('Json saved!')

ocs = oaklandCrimeStatistics(data_file_path='./Oakland Crime Statistics 2011 to 2016',
                             result_path='./fpar_results')
ocs.mining(min_support = 0.1, min_confidence = 0.5,head_n=50000)
```

```
'''
Visualization
'''
# boxplot of frequent itemset
with open('./fpar_results/freq.json') as f1:
    freq = [json.loads(each) for each in f1.readlines()]
    freq_sup = [each['sup'] for each in freq]
    plt.figure()
    plt.title('Frequent Itemset')
    plt.boxplot(freq_sup)
    plt.show()
    fig_name = './figs/freq.jpg'
    plt.savefig(fig_name)
# scatter of rules
with open('./fpar_results/rules.json') as f2:
    rules = [json.loads(each) for each in f2.readlines()]
    rules_sup = [each['sup'] for each in rules]
    rules_conf = [each['conf'] for each in rules]
    fig=plt.figure('rule')
    ax=fig.add_axes([0.1,0.1,0.8,0.8])
    ax.set_title('Rules')
    ax.scatter(rules_sup, rules_conf, marker='o', color='red')
    ax.set_xlabel('Support')
    ax.set_ylabel('Confidence')
    plt.show()
    fig_name = './figs/rules.jpg'
    plt.savefig(fig_name)
print('Visualization completed!')
```

结果与分析

数据集合并后的数据摘要

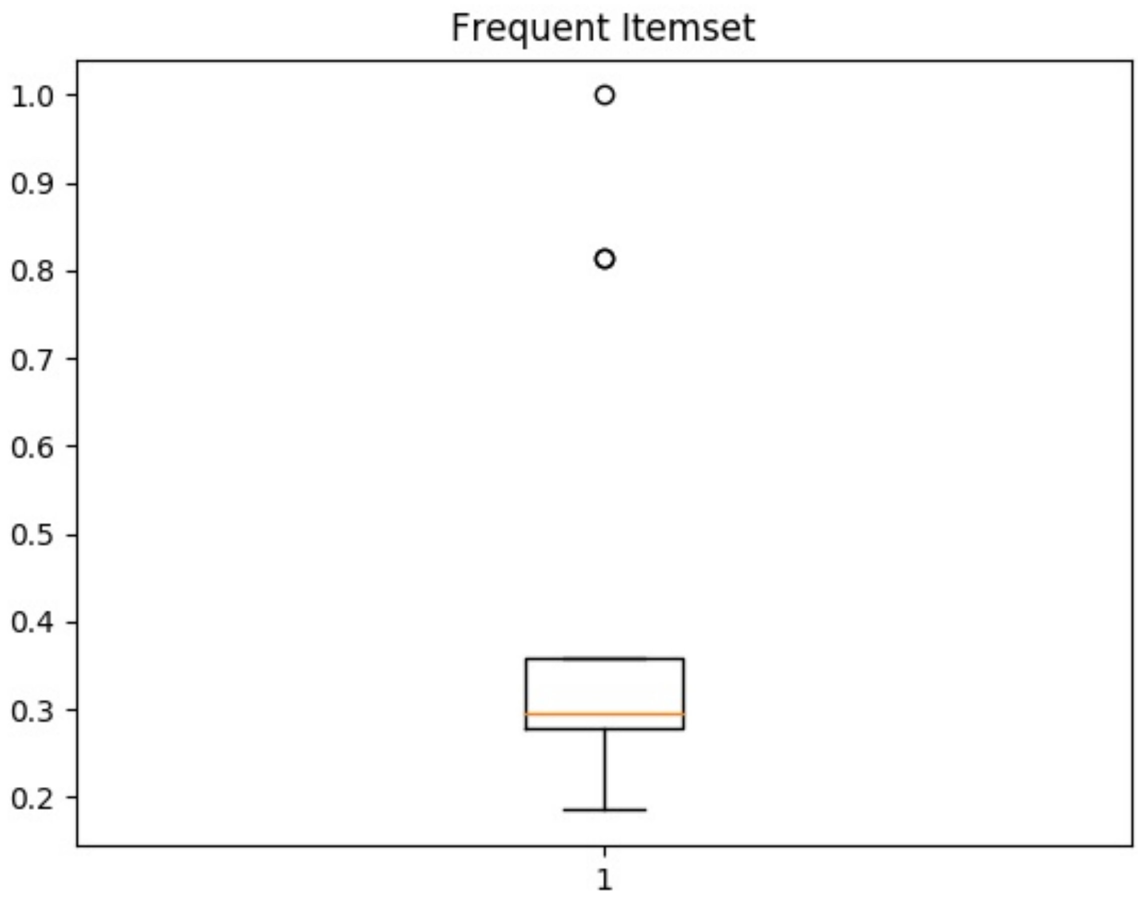
```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1046388 entries, 0 to 110827
Data columns (total 8 columns):
Agency                      1046384 non-null object
Location                    1046276 non-null object
Area Id                     864023 non-null object
Beat                        1040583 non-null object
Priority                    1046384 non-null float64
Incident Type Id            1046384 non-null object
Incident Type Description   1045996 non-null object
Event Number                1046384 non-null object
dtypes: float64(1), object(7)
memory usage: 71.8+ MB
None
```

频繁项集

将frequent itemset按照sup的值降序输出：

```
{'set': [('Agency', u'OP')], 'sup': 1.0}
{'set': [('Agency', u'OP'), ('Priority', 2.0)], 'sup': 0.81442}
{'set': [('Priority', 2.0)], 'sup': 0.81442}
{'set': [('Area Id', 1.0)], 'sup': 0.35754}
{'set': [('Agency', u'OP'), ('Area Id', 1.0)], 'sup': 0.35754}
{'set': [('Agency', u'OP'), ('Area Id', 3.0)], 'sup': 0.35092}
{'set': [('Area Id', 3.0)], 'sup': 0.35092}
{'set': [('Agency', u'OP'), ('Priority', 2.0), ('Area Id', 1.0)], 'sup': 0.29566}
{'set': [('Area Id', 1.0), ('Priority', 2.0)], 'sup': 0.29566}
{'set': [('Agency', u'OP'), ('Area Id', 2.0)], 'sup': 0.29154}
{'set': [('Area Id', 2.0)], 'sup': 0.29154}
{'set': [('Agency', u'OP'), ('Priority', 2.0), ('Area Id', 3.0)], 'sup': 0.27902}
{'set': [('Priority', 2.0), ('Area Id', 3.0)], 'sup': 0.27902}
{'set': [('Area Id', 2.0), ('Priority', 2.0)], 'sup': 0.23974}
{'set': [('Agency', u'OP'), ('Area Id', 2.0), ('Priority', 2.0)], 'sup': 0.23974}
{'set': [('Priority', 1.0)], 'sup': 0.18556}
{'set': [('Priority', 1.0), ('Agency', u'OP')], 'sup': 0.18556}
```

可视化



分析

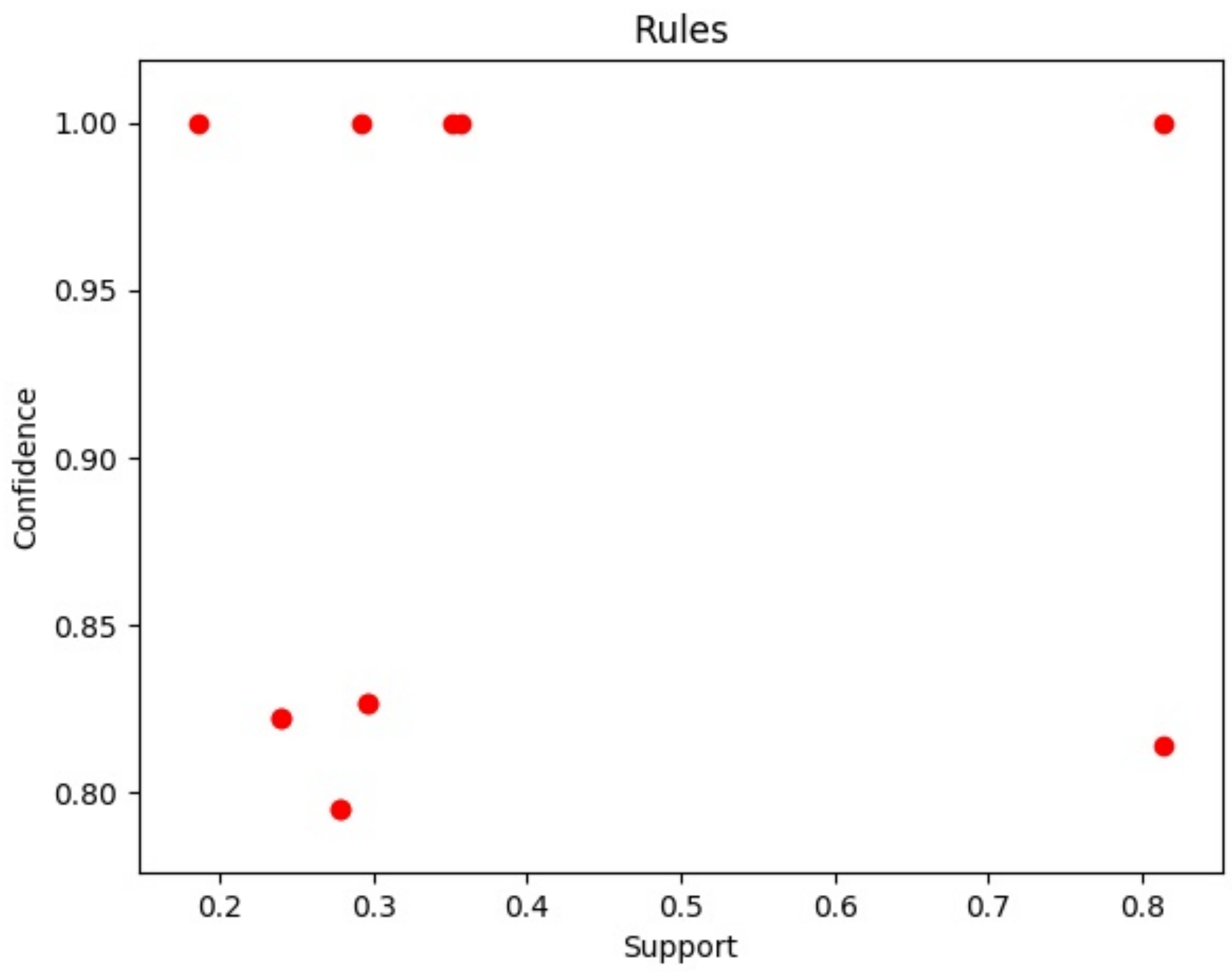
('Agency', u'OP')、('Priority', 2.0)、('Area Id', 1.0)以及它们的组合出现频率较高

关联规则

将rules按照conf的值降序输出：

```
{'X_set': [('Area Id', 1.0)], 'Jaccard': 0.35754, 'Y_set': [('Agency', u'OP')], 'lift':
2.796889858477373, 'conf': 1.0, 'sup': 0.35754}
{'X_set': [('Area Id', 2.0)], 'Jaccard': 0.29154000000000001, 'Y_set': [('Agency',
u'OP')], 'lift': 3.4300610550867803, 'conf': 1.0, 'sup': 0.29154}
{'X_set': [('Area Id', 3.0)], 'Jaccard': 0.35092000000000007, 'Y_set': [('Agency',
u'OP')], 'lift': 2.8496523424142253, 'conf': 1.0, 'sup': 0.35092}
{'X_set': [('Priority', 1.0)], 'Jaccard': 0.18556, 'Y_set': [('Agency', u'OP')],
'lift': 5.389092476826902, 'conf': 1.0, 'sup': 0.18556}
{'X_set': [('Priority', 2.0)], 'Jaccard': 0.81442, 'Y_set': [('Agency', u'OP')],
'lift': 1.2278676849782666, 'conf': 1.0, 'sup': 0.81442}
{'X_set': [('Area Id', 1.0)], 'Jaccard': 0.33739586899463647, 'Y_set': [('Priority',
2.0)], 'lift': 2.312827811034905, 'conf': 0.82692845555742, 'sup': 0.29566}
{'X_set': [('Area Id', 1.0)], 'Jaccard': 0.33739586899463647, 'Y_set': [('Agency',
u'OP'), ('Priority', 2.0)], 'lift': 2.312827811034905, 'conf': 0.82692845555742, 'sup':
0.29566}
{'X_set': [('Area Id', 2.0)], 'Jaccard': 0.2767657177160537, 'Y_set': [('Priority',
2.0)], 'lift': 2.8206175390907067, 'conf': 0.8223228373465047, 'sup': 0.23974}
{'X_set': [('Area Id', 2.0)], 'Jaccard': 0.2767657177160537, 'Y_set': [('Agency',
u'OP'), ('Priority', 2.0)], 'lift': 2.8206175390907067, 'conf': 0.8223228373465047,
'sup': 0.23974}
{'X_set': [('Agency', u'OP')], 'Jaccard': 0.81442, 'Y_set': [('Priority', 2.0)],
'lift': 0.81442, 'conf': 0.81442, 'sup': 0.81442}
{'X_set': [('Area Id', 3.0)], 'Jaccard': 0.3148072930769925, 'Y_set': [('Priority',
2.0)], 'lift': 2.2657870642323523, 'conf': 0.7951099965804171, 'sup': 0.27902}
{'X_set': [('Area Id', 3.0)], 'Jaccard': 0.3148072930769925, 'Y_set': [('Agency',
u'OP'), ('Priority', 2.0)], 'lift': 2.2657870642323523, 'conf': 0.7951099965804171,
'sup': 0.27902}
```

可视化



分析

Area Id的置信度较高，即犯罪记录的分布呈现区域性的特征。Area Id和Priority有较高的关联度，即犯罪地点和犯罪级别有关联性。