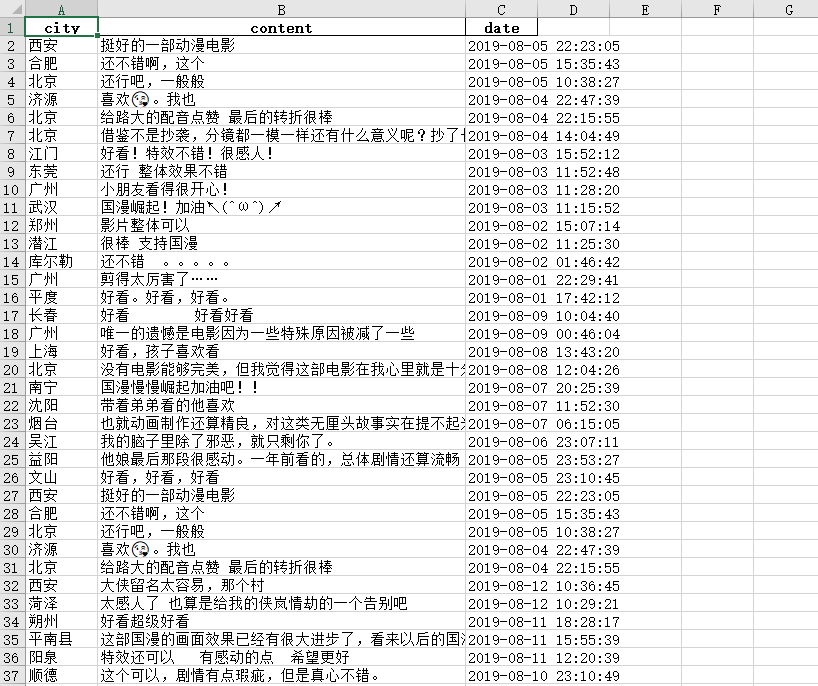
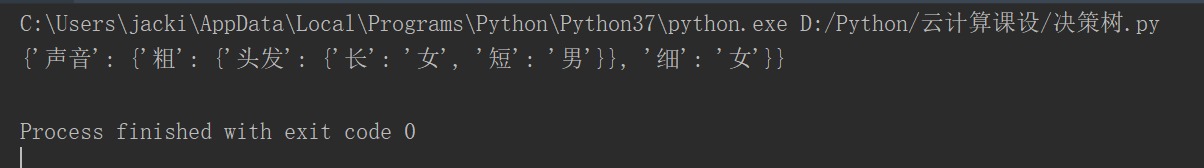
58二手车：



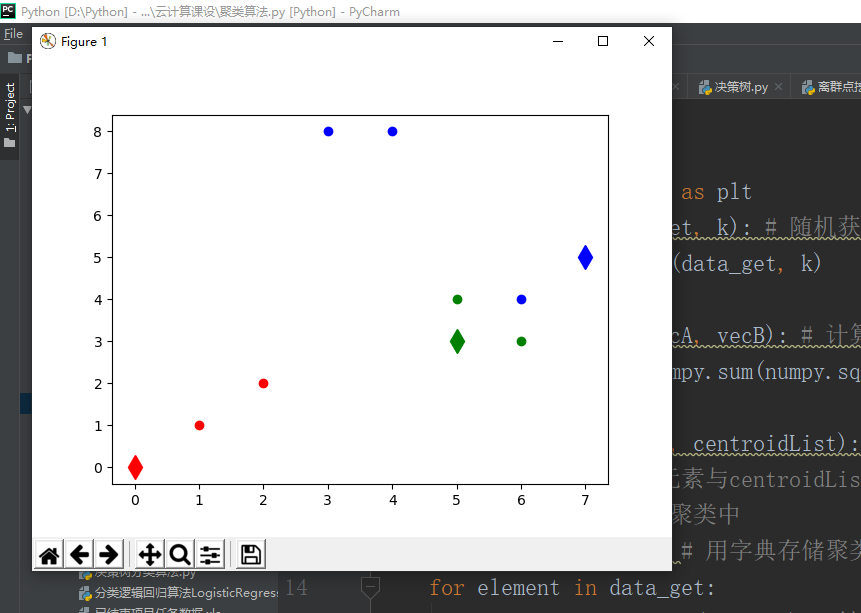
风雨骤：



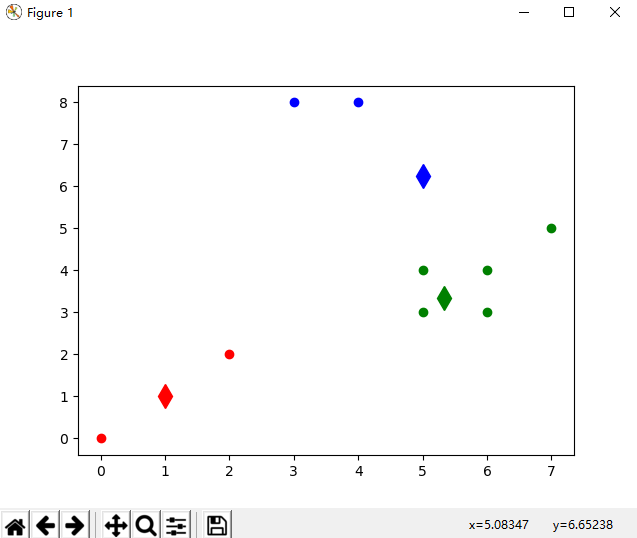
决策树：



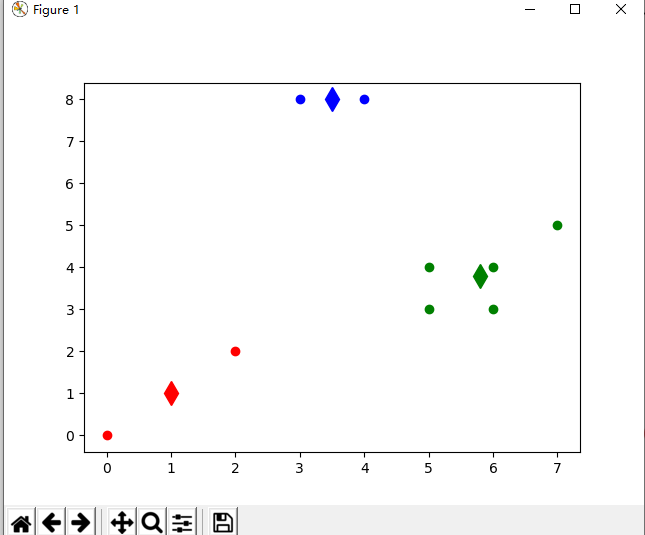
聚类算法：第一次聚类



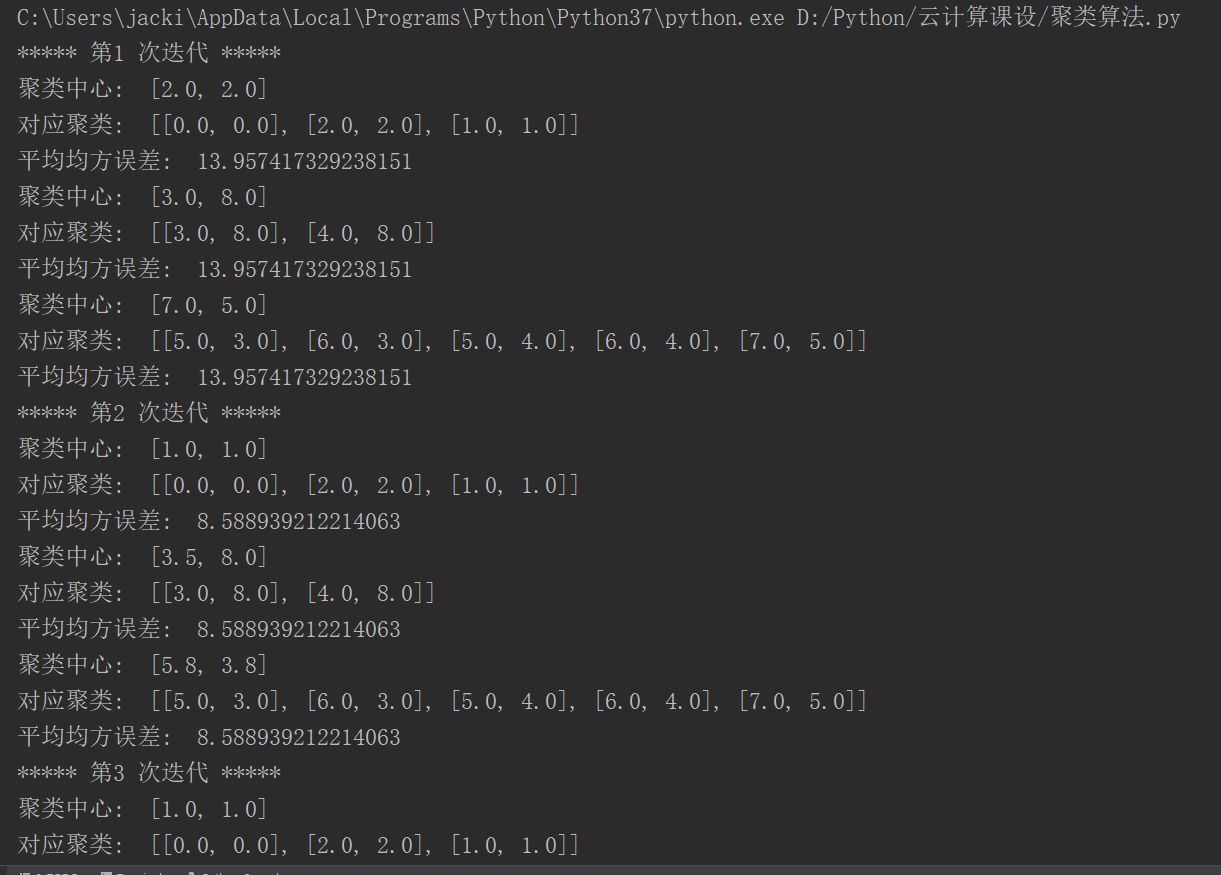
第二次聚类：



第三次聚类：

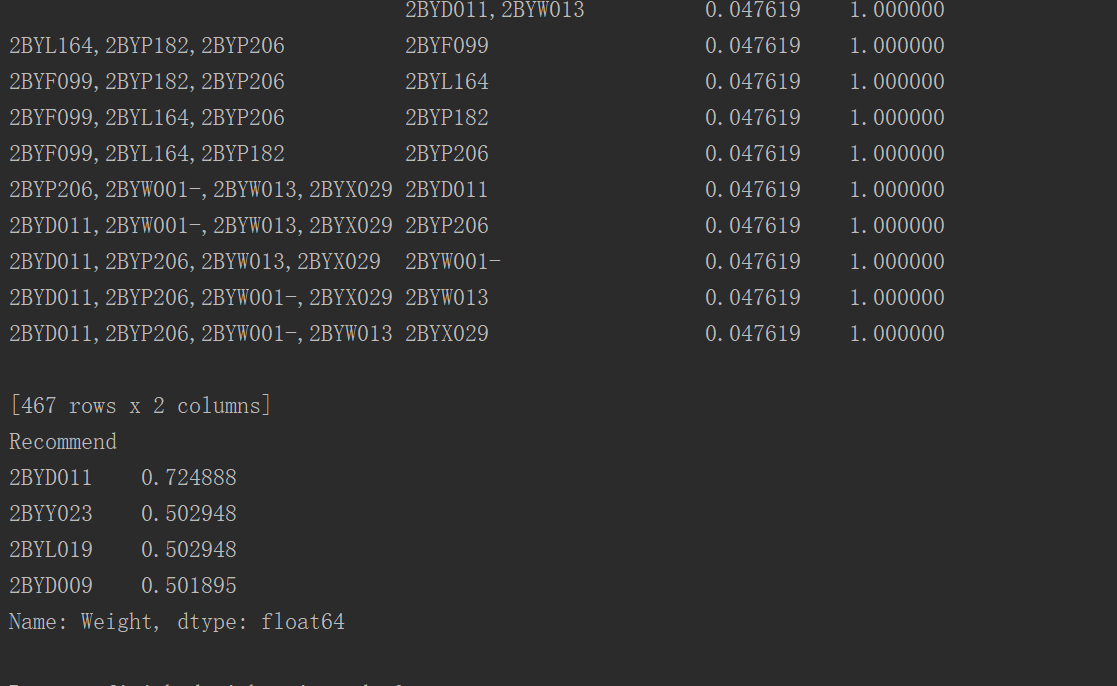


总结果：

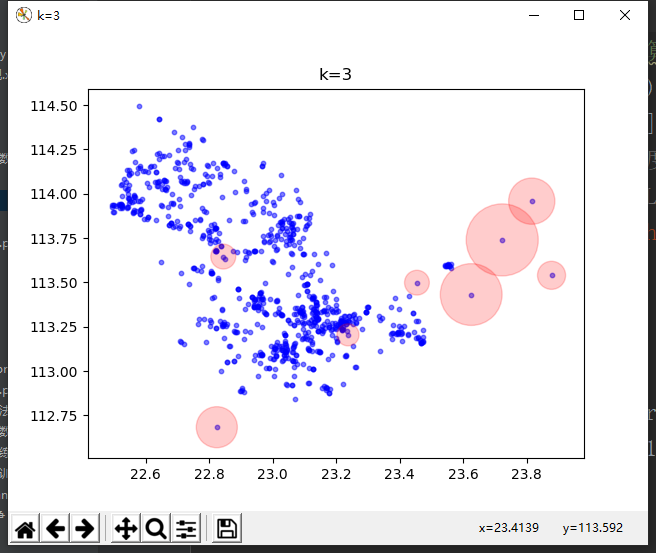


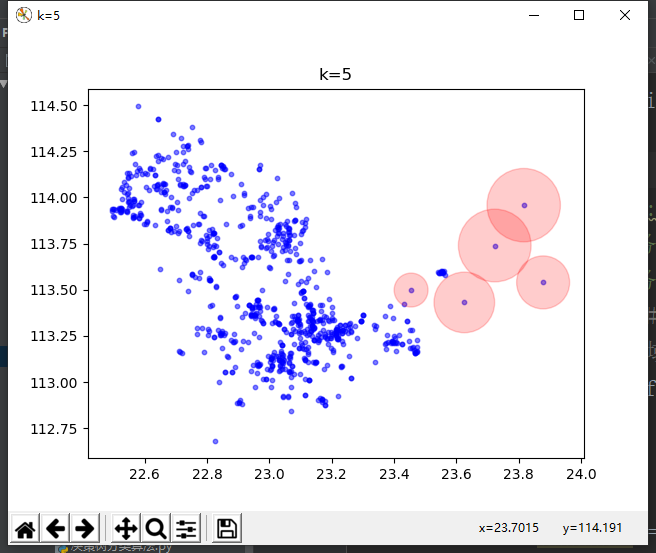
关联规则：

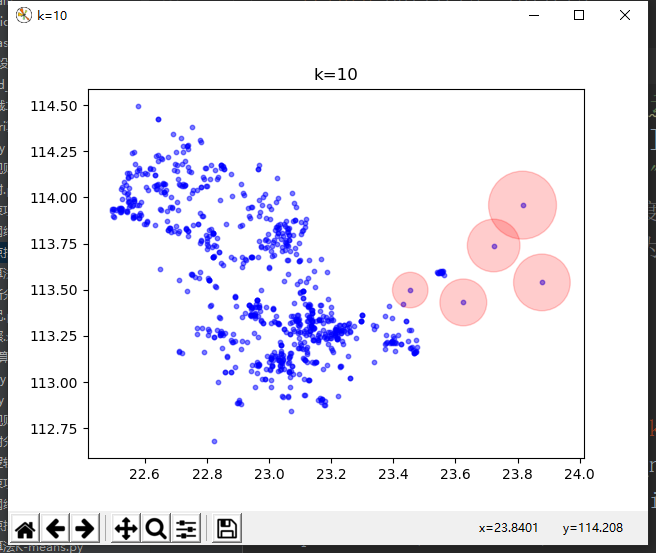
import os,itertools  
import numpy as np  
import pandas as pd  
class Apriori(object):  
 def \_\_init\_\_(self, itemSets, minSupport=0.5, minConf=0.7, sort = False):  
 self.itemSets = itemSets  
 self.minSupport = minSupport  
 self.minConf = minConf  
 self.sort = sort  
 self.\_\_Initialize()  
 def \_\_Initialize(self):  
 self.\_\_item()  
 self.\_\_creat\_matrix()  
 self.update(minSupport=self.minSupport, minConf=self.minConf)  
 def \_\_item(self):  
 *''' 获取项目元素列表 '''* self.item = []  
 for itemSet in self.itemSets:  
 for item in itemSet:  
 if item not in self.item:  
 self.item.append(item)  
 self.item.sort()  
 def \_\_creat\_matrix(self):  
 *''' 将项集转为 pandas.DataFrame 数据类型 '''* self.data = pd.DataFrame(columns=self.item)  
 for i in range(len(self.itemSets)):  
 self.data.loc[i, self.itemSets[i]] = 1  
 def \_\_candidate\_itemsets\_l1(self):  
 *''' 创建单项频繁项集及 L1'''* self.L1 = self.data.loc[:, self.data.sum(axis=0) / len(self.itemSets) >= self.minSupport]  
 self.L1\_support\_selects = dict(self.L1.sum(axis=0) / len(self.itemSets)) # 只作为分母，不进行拆分  
 def \_\_candidate\_itemsets\_lk(self):  
 *''' 根据 L1 创建多项频繁项集 Lk ，非频繁项集的任何超集都不是频繁项集 '''* last\_support\_selects = self.L1\_support\_selects.copy() # 初始化  
 while last\_support\_selects:  
 new\_support\_selects = {}  
 for last\_support\_select in last\_support\_selects.keys():  
 for L1\_support\_name in set(self.L1.columns) - set(last\_support\_select.split(',')):  
 columns = sorted([L1\_support\_name] + last\_support\_select.split(',')) # 新的列名：合并后排序  
 count = (self.L1.loc[:, columns].sum(axis=1) == len(columns)).sum()  
 if count / len(self.itemSets) >= self.minSupport:  
 new\_support\_selects[','.join(columns)] = count / len(self.itemSets)  
 self.support\_selects.update(new\_support\_selects)  
 last\_support\_selects = new\_support\_selects.copy() # 作为新的 Lk，进行下一轮更新  
 def \_\_support\_selects(self):  
 *''' 支持度选择 '''* self.\_\_candidate\_itemsets\_l1()  
 self.\_\_candidate\_itemsets\_lk()  
 self.item\_Conf = self.L1\_support\_selects.copy()  
 self.item\_Conf.update(self.support\_selects)  
 def \_\_confidence\_selects(self):  
 *''' 生成关联规则，其中 support\_selects 已经按照长度大小排列 '''* for groups, Supp\_groups in self.support\_selects.items():  
 groups\_list = groups.split(',')  
 for recommend\_len in range(1, len(groups\_list)):  
 for recommend in itertools.combinations(groups\_list, recommend\_len):  
 items = ','.join(sorted(set(groups\_list) - set(recommend)))  
 Conf = Supp\_groups / self.item\_Conf[items]  
 if Conf >= self.minConf:  
 self.confidence\_select.setdefault(items, {})  
 self.confidence\_select[items].setdefault(','.join(recommend),{'Support': Supp\_groups,  
'Confidence': Conf})  
 def show(self,\*\*kwargs):  
 *''' 可视化输出 '''* if kwargs.get('data'):  
 select = kwargs['data']  
 else:  
 select = self.confidence\_select  
 items = []  
 value = []  
 for ks, vs in select.items():  
 items.extend(list(zip([ks] \* vs.\_\_len\_\_(), vs.keys())))  
 for v in vs.values():  
 value.append([v['Support'], v['Confidence']])  
 index = pd.MultiIndex.from\_tuples(items, names=['Items', 'Recommend'])  
 self.rules = pd.DataFrame(value, index=index, columns=['Support', 'Confidence'])  
 if self.sort or kwargs.get('sort'):  
 result = self.rules.sort\_values(by=['Support', 'Confidence'], ascending=False)  
 else:  
 result = self.rules.copy()  
 return result  
 def update(self, \*\*kwargs):  
 *''' 用于更新数据 '''* if kwargs.get('minSupport'):  
 self.minSupport = kwargs['minSupport']  
 self.support\_selects = {} # 用于储存满足支持度的频繁项集  
 self.\_\_support\_selects()  
 if kwargs.get('minConf'):  
 self.minConf = kwargs['minConf']  
 self.confidence\_select = {} # 用于储存满足自信度的关联规则  
 self.\_\_confidence\_selects()  
 print(self.show())  
 if kwargs.get('file\_name'):  
 file\_name = kwargs['file\_name']  
 self.show().to\_excel(f'/../table/{file\_name}.xlsx')  
 self.apriori\_rules = self.rules.copy()  
 def \_\_get\_Recommend\_list(self,itemSet):  
 *''' 输入数据，获取关联规则列表 '''* self.recommend\_selects = {}  
 itemSet = set(itemSet) & set(self.apriori\_rules.index.levels[0])  
 if itemSet:  
 for start\_str in itemSet:  
 for end\_str in self.apriori\_rules.loc[start\_str].index:  
 start\_list = start\_str.split(',')  
 end\_list = end\_str.split(',')  
 self.\_\_creat\_Recommend\_list(start\_list, end\_list, itemSet)  
 def \_\_creat\_Recommend\_list(self,start\_list,end\_list,itemSet):  
 *''' 迭代创建关联规则列表 '''* if set(end\_list).issubset(itemSet):  
 start\_str = ','.join(sorted(start\_list+end\_list))  
 if start\_str in self.apriori\_rules.index.levels[0]:  
 for end\_str in self.apriori\_rules.loc[start\_str].index:  
 start\_list = start\_str.split(',')  
 end\_list = end\_str.split(',')  
 self.\_\_creat\_Recommend\_list(sorted(start\_list),end\_list,itemSet)  
 elif not set(end\_list) & itemSet:  
 start\_str = ','.join(start\_list)  
 end\_str = ','.join(end\_list)  
 self.recommend\_selects.setdefault(start\_str, {})  
 self.recommend\_selects[start\_str].setdefault(end\_str, {'Support': self.apriori\_rules.loc[(start\_str,  
end\_str), 'Support'], 'Confidence': self.apriori\_rules.loc[(start\_str, end\_str), 'Confidence']})  
 def get\_Recommend(self,itemSet,\*\*kwargs):  
 *''' 获取加权关联规则 '''* self.recommend = {}  
 self.\_\_get\_Recommend\_list(itemSet)  
 self.show(data = self.recommend\_selects)  
 items = self.rules.index.levels[0]  
 for item\_str in items:  
 for recommends\_str in self.rules.loc[item\_str].index:  
 recommends\_list = recommends\_str.split(',')  
 for recommend\_str in recommends\_list:  
 self.recommend.setdefault(recommend\_str,0)  
 self.recommend[recommend\_str] += self.rules.loc[(item\_str,recommends\_str),'Support'] \* self.rules.loc[(item\_str,recommends\_str),'Confidence'] \* self.rules.loc[item\_str,'Support'].mean()/(self.rules.loc[item\_str,'Support'].sum()\*len(recommends\_list))  
 result = pd.Series(self.recommend,name='Weight').sort\_values(ascending=False)  
 result.index.name = 'Recommend'  
 result = result/result.sum()  
 result = 1/(1+np.exp(-result))  
 print(result)  
 if kwargs.get('file\_name'):  
 file\_name = kwargs['file\_name']  
 excel\_writer = pd.ExcelWriter(f'{os.getcwd()}/../table/{file\_name}.xlsx')  
 result.to\_excel(excel\_writer,'推荐项目及权重')  
 self.rules.to\_excel(excel\_writer, '关联规则树状表')  
 self.show().to\_excel(excel\_writer, '总关联规则树状表')  
 self.show(sort = True).to\_excel(excel\_writer, '总关联规则排序表')  
 excel\_writer.save()  
 return result  
def str2itemsets(strings, split=','):  
 *''' 将字符串列表转化为对应的集合 '''* itemsets = []  
 for string in strings:  
 itemsets.append(sorted(string.split(split)))  
 return itemsets  
if \_\_name\_\_ == '\_\_main\_\_':  
 # 1.导入数据  
 data = pd.read\_excel(r'apriori算法实现.xlsx', index=False)  
 # 2.关联规则中不考虑多次购买同一件物品，删除重复数据  
 data = data.drop\_duplicates()  
 # 3.初始化列表  
 itemSets = []  
 # 3.按销售单分组，只有 1 件商品的没有意义，需要进行过滤  
 groups = data.groupby(by='销售单明细')  
 for group in groups:  
 if len(group[1]) >= 2:  
 itemSets.append(group[1]['商品编码'].tolist())  
 # 4.训练 Apriori  
 ap = Apriori(itemSets, minSupport=0.03, minConf=0.5)  
 ap.get\_Recommend('2BYP206,2BYW001-,2BYW013,2BYX029'.split(','))



离群点挖掘：







神经网络：

