数据挖掘实验教程

A Guide to Experiments of Data Mining



2023

西南财经大学计算机与人工智能学院

邱江涛



目录

[第一章：概述 3](#_Toc136789548)

[第二章：数据挖掘实验环境配置 4](#_Toc136789549)

[2.1 本地平台的配置 4](#_Toc136789550)

[第三章：数据预处理实验 5](#_Toc136789551)

[3.1 绘图中显示中文 5](#_Toc136789552)

[第四章：数据探索分析实验 7](#_Toc136789553)

[第五章：分类实验 8](#_Toc136789554)

[5.1 处理混合类型的数据集 8](#_Toc136789555)

[5.2 在混合类型数据集上建立SVM分类模型 13](#_Toc136789556)

[5.3 基于网格搜索的超参数选择 14](#_Toc136789557)

[第六章：回归实验 15](#_Toc136789558)

[6.1 在混合类型数据集上的回归模型 15](#_Toc136789559)

[第七章：聚类实验 17](#_Toc136789560)

[7.1 聚类趋势分析 17](#_Toc136789561)

[7.2 理解聚类结果 19](#_Toc136789562)

[7.3 DBSCAN的参数选取 19](#_Toc136789563)

[7.4 案例：客户细分 19](#_Toc136789564)

[第八章：特征工程实验 27](#_Toc136789565)

[8.1 特征选择 27](#_Toc136789566)

[8.2 特征工程工具 27](#_Toc136789567)

[第九章：关联规则分析：零售业的分析工具 28](#_Toc136789568)

[9.1 最大频繁项集和频繁闭项集 28](#_Toc136789569)

[9.2 零售业的营销策略 29](#_Toc136789570)

# 第一章：概述

数据挖掘课程是

# 第二章：数据挖掘实验环境配置

现在的python开发环境的选择种类非常多。即可以在本地安装python和IDE。也可以在使用一些云平台提供的网上开发环境，例如，国内的阿里天池Notebook和百度的AI Studio，还有很多专门为深度学习提供GPU云平台，如AutoDL, 恒源云等。国外的云平台以谷歌的Colaborotary最为著名。

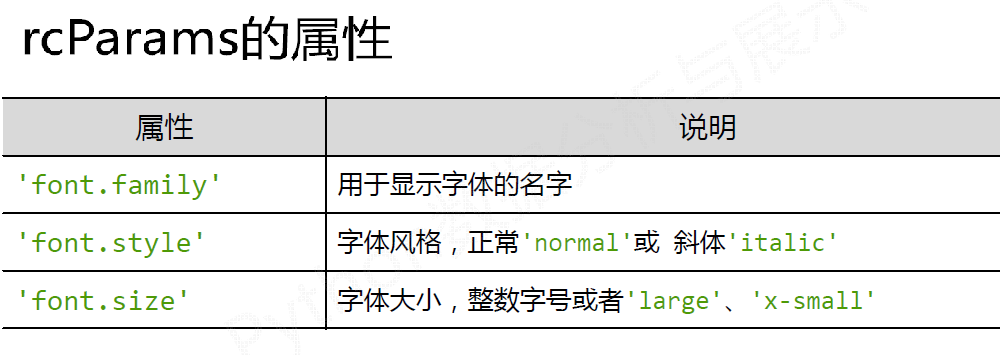
下面分别介绍这些环境下的数据挖掘实验平台的搭建和配置。

## 2.1 本地平台的配置

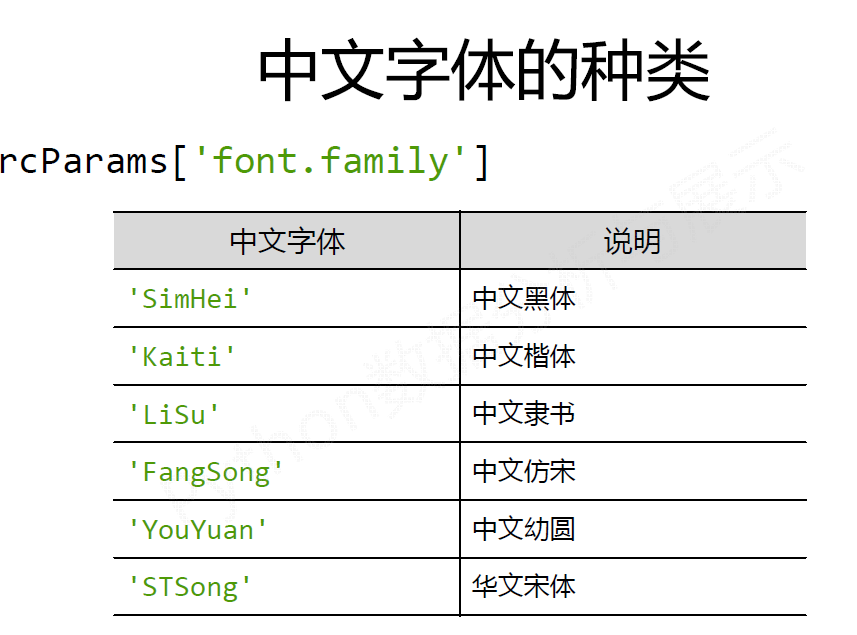
# 第三章：数据预处理实验

## 3.1 绘图中显示中文

Pyplot并不默认支持中文显示，需要rcParams修改成中文字体实现。



支持的中文字体有



所以设置成中文字体即可，例如

plt.rcParams['font.family'] = 'SimHei'

代码3-1：显示中文

|  |
| --- |
| import pickle  import pandas as pd  import numpy as np  import matplotlib.pyplot as plt  plt.rcParams['font.sans-serif']=['SimHei']  f=open('economic.pl', 'rb')  data = pickle.load(f)  f.close()  plt.figure(figsize=(8,7))## 设置画布  plt.scatter(data.iloc[:,0],data.iloc[:,2], marker='o')## 绘制散点图  plt.xlabel('年份')## 添加横轴标签  plt.ylabel('生产总值（亿元）')## 添加y轴名称  plt.xticks(range(0,70,4),data.iloc[range(0,70,4),1],rotation=45)  plt.title('2000-2017年季度生产总值散点图')## 添加图表标题 |

# 第四章：数据探索分析实验

# 第五章：分类实验

## 5.1 处理混合类型的数据集

完成数据挖掘任务前，需要先处理数据集时往往会遇到混合类型的数据。一个简单的思路是：首先把不同类型的属性区分开，然后单独的进行处理，再合并创建一个统一的数据集。下面介绍几个例子，它们对分类属性进行one-hot编码处理，对数值属性进行规范化。重要的是，用**ColumnTransformer**将它们集中处理，非常简便。

1. **sklearn.preprocessing.OneHotEncoder**

对类别属性进行one-hot编码

class sklearn.preprocessing.OneHotEncoder(\*, categories='auto', drop=None, sparse='deprecated', sparse\_output=True, dtype=<class 'numpy.float64'>, handle\_unknown='error', min\_frequency=None, max\_categories=None)[source]

主要参数如下：

（1）categories：‘auto’或者一个属性列表, default=’auto’

（2）drop：{‘first’, ‘if\_binary’} or an array-like of shape (n\_features,), default=None

设定一种方法来丢弃one-hot编码后产生的新的属性中的某一个。因为存在完全多重共线性，可能会导致后面的任务出现偏差。有下面三种选择：

None：默认值，不删除。

‘first’：丢弃第一个新属性。

‘if\_binary’：如果原始属性有两个属性值，丢弃第一个属性值。其余情况，即原始属性有两个以上的属性值，保留所有新属性完整。

（3）sparse\_output：bool值，默认为True。为True时返回稀疏矩阵，否则返回数组

（4）handle\_unknown：遇到未知值的处理方法，有三种选择

‘error’：产生出错信息，这是默认值。

‘ignore’ : 设置为0。

‘infrequent\_if\_exist’：设置成最不频繁的少数属性值。

常用的两个函数是：

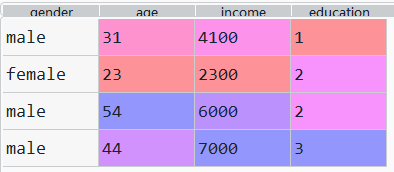
* fit(X[, y])：用一个数据集X去拟合一个OneHotEncoder的对象。其含义是，一个OneHotEncoder数据对象再数据集上拟合是在学习分类属性的特征，包括属性值等信息。然后就可以处理具有相同属性的其他数据集。
* transform(X)：在OneHotEncoder的对象拟合了一个数据集后，使用该方法对数据集X进行one-hot编码。返回的结果是ndarray的spare matrix数据对象。
* fit\_transform(X[, y])：用一个数据集X去拟合一个OneHotEncoder的对象，并把X进行one-hot编码。fit\_transform函数看作是把fit和transform两个函数集成到了一起。

使用OneHotEncoder的基本步骤是，

* 首先准备好要处理的数据集，它可以是numpy或pandas的数据对象。它应该都是类别属性。如果包含数值属性，会被当做类别属性处理。
* 创建一个OneHotEncoder的对象。后面对数据集进行one-hot编码时，就是使用该对象。
* 用创建的OneHotEncoder的对象去拟合一个数据集。其目的是了解一个数据集的类别属性和属性值。然后，后面就可以用改对象，去对其他相同结果的数据集进行one-hot编码。一个典型的应用场景是在对训练集中的类别属性进行拟合，然后可以对测试集进行相同的处理。拟合可以用fit函数，或fit\_transform函数。
* 用拟合后的OneHotEncoder的对象去处理数据集。采用transform函数。当然fit\_transform函数可以拟合完成后就转换数据集。

**练习5.1：基本的one-hot编码**

本节使用一个简单的职工信息数据集faculty.csv，只有四条记录。



其中属性gender和education是分类属性。属性age和income是数值属性。下面使用OneHotEncoder对分类属性进行one-hot编码时，需要单独对分类属性进行处理，例如：

|  |
| --- |
| import pandas as pd  import numpy as np  from sklearn.preprocessing import OneHotEncoder  df = pd.read\_csv('data/faculty.csv')  cols = ['gender','education']  df2 = df[cols]  enc = OneHotEncoder()  enc.fit(df2)  X=enc.transform(df2).toarray() |

上述代码创建了一个OneHotEncoder对象，然后在只有分类属性的数据帧df2上拟合数据。再将该数据帧转换成ndarray数据结构。

当然，上面拟合和转换是两个步骤。也可以用fit\_transform函数一次性的拟合和转换。其返回的结果是一个scipy的sparse matrics数据结构。调用其toarry()函数可以转换成numpy的ndarray数据结构。

从上面的操作可以看出，如果要训练一个模型，还需要把得到的one-hot编码后的X和原先数据帧中的数值属性进行合并。这样的操作比较复杂，下面再介绍几个函数，通过创建pipeline函数可以方便进行数据集处理的操作。

1. **sklearn.compose.** **ColumnTransformer**

创建一个可以应用在pandas数据帧或者numpy的array数据集上的部分列上的转换器（Transformer）。它非常适合处理混合类型数据的数据集。

class sklearn.compose.ColumnTransformer(transformers, \*, remainder='drop', sparse\_threshold=0.3, n\_jobs=None, transformer\_weights=None, verbose=False, verbose\_feature\_names\_out=True)

创建转换器对象时，它的参数是包含多个元组的列表。每个元组代表一种处理，它的内容是

(name, transformer, columns)

Name是给当前的处理分配的名称

Transformers 当前的处理器

Columns表示在哪些列上处理，它是一个列名的列表。

和OneHotEncoder一样**ColumnTransformer**有三个主要方法。具体内容和OneHotEncoder一样。

fit(X[, y])，transform(X)，fit\_transform(X[, y])

**练习5.2： 使用ColumnTransformer处理混合类型数据集**

将练习5.1中的数据集数值属性进行规范化，类别属性进行one-hot编码。

|  |
| --- |
| import pandas as pd  import numpy as np  from sklearn.compose import ColumnTransformer  from sklearn.preprocessing import OneHotEncoder, StandardScaler  from sklearn.pipeline import make\_pipeline  from sklearn.preprocessing import OneHotEncoder  df = pd.read\_csv('data/faculty.csv')  numeric\_features = ["age", "income"]  categorical\_features = ['gender','education']  preprocessor = ColumnTransformer(  [  ("num", StandardScaler(),numeric\_features),  ("cat",  OneHotEncoder(handle\_unknown="ignore"),  categorical\_features,  ),  ],  verbose\_feature\_names\_out=False,  )  X=preprocessor.fit\_transform(df) |

上面的程序创建了一个ColumnTransformer类的对象preprocessor。它包括了两个属性的处理：对数值属性进行零均值规范化，对类别属性one-hot编码。

1. **自动按照属性的类型分别进行不同的预处理**

例子5.2中，预先有用户给出了哪些属性是数值型，哪些是类别型。如果属性很多时如此会比较麻烦。此时可以用make\_column\_selector函数，根据正则表达式自动选择不同属性。

**sklearn.compose.make\_column\_selector** sklearn.compose.make\_column\_selector(pattern=None, \*, dtype\_include=None, dtype\_exclude=None)

patter是正则表达式，设定按照属性的名词来信选择属性。如果不给出，则不会基于正则表达式来选择属性。

dtype\_include：按照属性类型来选择一些属性

dtype\_exclude：按照属性类型来排除一些属性

在使用时，该函数通常作为创建ColumnTransformer类的对象时的参数。

**练习5.3：使用ColumnTransformer和make\_column\_selector处理混合类型数据集**

下面代码把make\_column\_selector嵌入到ColumnTransformer类创建对象的过程中参数，自动根据属性类型来选择属性。

|  |
| --- |
| import pandas as pd  import numpy as np  from sklearn.compose import ColumnTransformer  from sklearn.preprocessing import OneHotEncoder, StandardScaler  from sklearn.pipeline import make\_pipeline  from sklearn.preprocessing import OneHotEncoder  from sklearn.compose import make\_column\_selector  df = pd.read\_csv('data/faculty.csv')  df['education'] = df['education'].astype(str)  preprocessor = ColumnTransformer(  [  ("num",  StandardScaler(),  **make\_column\_selector**(dtype\_include=np.number)),  ("cat",  OneHotEncoder(handle\_unknown="ignore"),  **make\_column\_selector**(dtype\_include=object),  ),  ]  )  X=preprocessor.fit\_transform(df) |

在练习5.1的数据集中，属性education应该是类别属性，但使用pandas的read\_csv读入时，自动转换成了数值属性。所以上面的代码首先给它转换会str类型。

df['education'] = df['education'].astype(str)

在创建ColumnTransformer的转换器时，其参数是用多个元组表示的多个处理步骤。每个处理步骤是一个元组，例如

("num", StandardScaler(), make\_column\_selector(dtype\_include=np.number)),

表示该处理步骤是用make\_column\_selector(dtype\_include=np.number)选择数值型的属性，进行零-均值规范化处理，该处理步骤的名词是num。

("cat", OneHotEncoder(handle\_unknown="ignore"), make\_column\_selector(dtype\_include=object) )

这一处理步骤将类型是object，即不是数值类型的都看作是类别属性。然后进行one-hot编码。其中handle\_unknown="ignore"表示，one-hot编码时，遇到不知道的属性值，则忽略。

## 5.2 在混合类型数据集上建立SVM分类模型

在《数据挖掘》教材的6.6.2节使用SVM在Universal Bank数据集上建立分类模型时，将类别属性用数值表示，将其看作了数值属性。本节将采用one-hot编码处理类别属性。并采用accuracy, balanced accuracy和F1指标评价模型性能。

**练习5.4： Universal Bank数据集的SVM模型**

Universal Bank数据集上有4个类别属性：'Securities Account', 'CD Account', 'Online', 'CreditCard'。它们将被one-hot编码。数值类型属性将被零-均值规范化。'Education'是序数属性，我们这里当作是数值属性。

|  |
| --- |
| import pandas as pd  import numpy as np  from sklearn.model\_selection import train\_test\_split  from sklearn.pipeline import make\_pipeline  from sklearn.preprocessing import StandardScaler  from sklearn.svm import SVC, NuSVC  from sklearn.compose import ColumnTransformer  from sklearn.compose import make\_column\_selector  from sklearn.preprocessing import OneHotEncoder  df=pd.read\_csv('data/UniversalBank.csv')  y = df['Personal Loan']  df = df.drop(['ID', 'ZIP Code','Personal Loan'], axis=1)  cat\_attr = ['Securities Account', 'CD Account', 'Online', 'CreditCard']  for attr in cat\_attr:  df[attr] = df[attr].astype(str)  preprocessor = ColumnTransformer(  [  ("num",  StandardScaler(),  make\_column\_selector(dtype\_include=np.number)),  ("cat",  OneHotEncoder(handle\_unknown="ignore"),  make\_column\_selector(dtype\_include=object),  ),  ]  )  X=preprocessor.fit\_transform(df)  X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=0)  model = SVC(gamma='auto', C=3, class\_weight={0:1,1:9})  model.fit(X\_train, y\_train)  y\_pred = model.predict(X\_test)  from sklearn.metrics import accuracy\_score  res = accuracy\_score(y\_test, y\_pred)  print('SVC模型的ACC:\n', res)  from sklearn.metrics import balanced\_accuracy\_score  res = balanced\_accuracy\_score(y\_test, y\_pred)  print('SVC模型的BAC:\n', res)  from sklearn.metrics import f1\_score  res = f1\_score(y\_test, y\_pred)  print('SVC模型的F1:\n', res) |

思考：

1. 将上面的实验结果和《数据挖掘》教材的6.6.2节的实验结果进行比较。能得出什么结论？
2. model = SVC(gamma='auto', C=3, class\_weight={0:1,1:9})语句中

class\_weight={0:1,1:9}是什么含义？

## 5.3 基于网格搜索的超参数选择

# 第六章：回归实验

## 6.1 在混合类型数据集上的回归模型

《数据挖掘》教材中没有给出MAE和MSE的评分计算函数。这里给出sklearn提供的评分计算函数。

1. **MAE**

sklearn.metrics.mean\_absolute\_error(y\_true, y\_pred)

主要参数介绍如下：

y\_true：真实目标值

y\_pred：模型预测结果

1. MSE

sklearn.metrics.mean\_squared\_error(y\_true, y\_pred, squared=True)

主要参数同MAE。其中

Squared是布尔值, 默认是True返回MSE，否则返回RMSE。默认是True

两个函数的使用例子如下：

|  |
| --- |
| from sklearn.metrics import mean\_squared\_error  y\_true = [3, -0.5, 2, 7]  y\_pred = [2.5, 0.0, 2, 8]  mean\_squared\_error(y\_true, y\_pred) |

**练习6.1： King County的房屋销售数据集上的回归模型**

美国的King County的房屋销售数据集包括了2014年五月到2015年五月期间的King County的房屋销售数据。它包括三个文件kc\_house\_train.csv是训练集；kc\_house\_test.csv是不包含价格的测试集；kc\_house\_label.csv是对应测试集中每条数据的房屋价格。数据集各个属性的描述见表5-1。

表5-1：King County的房屋销售数据集的属性描述

|  |  |  |
| --- | --- | --- |
| 序号 | 属性名 | 解释 |
| 1 | date | 房屋销售日期 |
| 2 | bedrooms | 卧室数量 |
| 3 | bathrooms | 浴室数量，0.5表示是无浴室厕所 |
| 4 | sqft\_living | 生活空间面积（平方英尺） |
| 5 | sqft\_lot | 室外空间面积（平方英尺） |
| 6 | floors | 房屋层数 |
| 7 | waterfront | 是否能看见海滨（0或1） |
| 8 | view | 房屋的视野（0-4星评分） |
| 9 | condition | 房屋的当前状况（1-5星评分） |
| 10 | grade | 针和condition不同的是，grade对公寓结构、构造、家具等给出的1-13星级评分。例如，1901年建的豪华房屋有12星的grade评分，但condition是1星。 |
| 11 | sqft\_above | 地面上部分的面积 |
| 12 | sqft\_basement | 地下室部分的面积 |
| 13 | yr\_built | 建造的年份 |
| 14 | yr\_renovated | 翻新的年份 |
| 15 | zipcode | 邮政编码 |
| 16 | lat | 维度 |
| 17 | long | 经度 |
| 18 | sqft\_living15 | 最近的15个邻居的平均生活空间面积（平方英尺） |
| 19 | sqft\_lot15 | 最近的15个邻居的平均室外空间面积（平方英尺） |
| 20 | Price | 房屋销售价格 |

数据集的预处理工作包括：

* date：针对日期数据，抽取其中的年份。一共有2014和2015两个年份，如此将该属性转换成一个0、1值的属性。
* yr\_built：将该属性转换，建造年份的最大值减去当前建造年份
* yr\_renovated：将该属性转换，翻新年份的最大值减去当前翻新年份
* zipcode：邮政编码能反映出房屋的区域位置。邮政编码是6位。当前数据集一共有70个独特值。取前4位作为区域划分，有19个独特值。将他们转换成类别属性。
* 删除了三个属性：id、lat、long

# 第七章：聚类实验

## 7.1 聚类趋势分析

在实施聚类任务前，首先需要分析待聚类的数据的分布。只有不是均匀分布的数据集，才有可能得到有意义的聚类结果，称之为簇。反过来说，在均匀分布的数据集上进行聚类操作即使得到了结果，但这个结果是随机的，不稳定的（每次实施聚类操作得到的结果差异很大）和无意义的。这个分析过程称为聚类趋势评估。

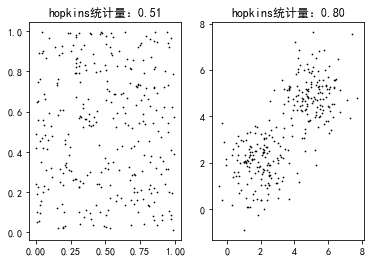


图7.1 数据均匀分布（A）和不均匀分布（B）

例如，图7.1（A）中数据的分布是均匀的，没有呈现聚类的趋势。聚类算法在该数据集上得不到一个合理、有意义的结果。图9.7（B）的数据则相对分布不均匀。

霍普金斯统计（Hopkins Statistic）是一种空间统计。它检验一个变量在空间分布的空间随机性。给定一个数据集D，它是随机变量o的样本。当想知道变量o是否被均匀的分布在数据空间，可以采用下面的步骤计算该变量的霍普金斯统计。

1. 从数据集D中随机的抽取n个样本，。对于每个样本找到它在数据集D中的最近邻。用表示和它的最近邻的距离。
2. 从数据集D中随机的抽取n个样本，。对于每个样本找到它在数据集{ D-Y }中的最近邻。用表示和它的最近邻的距离。
3. 按照公式（9.13）计算计算霍普金斯统计量H

如果数据集D是均匀分布的，则的计算值应该接近。H值应该在0.5附近。如果D展现了聚类特性，随机抽取的n个数据应该表现聚集特性，它们的应该比小。即计算的H应该大于0.5。且聚类趋势越明显，H值越大。当H>0.75时，表示以90%的置信水平详细存在这聚类趋势。图7.1两个数据集计算Hopkins统计的代码和结果如下：

代码7-1：聚类趋势分析

|  |
| --- |
| import numpy as np  import matplotlib.pyplot as plt  from random import sample  from numpy.random import uniform  from sklearn.neighbors import NearestNeighbors  # 定义hopkins统计量函数  def hopkins\_statistic(X):  sample\_size = int(X.shape[0]\*0.05)  X\_uniform\_random\_sample = uniform(X.min(axis=0), X.max(axis=0) ,(sample\_size , X.shape[1]))  random\_indices=sample(range(0, X.shape[0], 1), sample\_size)  X\_sample = X[random\_indices]  neigh = NearestNeighbors(n\_neighbors=2)  nbrs=neigh.fit(X)  u\_distances , u\_indices = nbrs.kneighbors(X\_uniform\_random\_sample , n\_neighbors=2)  u\_distances = u\_distances[: , 0] #distance to the first (nearest) neighbour  w\_distances , w\_indices = nbrs.kneighbors(X\_sample , n\_neighbors=2)  w\_distances = w\_distances[: , 1]  u\_sum = np.sum(u\_distances)  w\_sum = np.sum(w\_distances)  H = u\_sum/ (u\_sum + w\_sum)  return H  # 数据集生成  X = np.random.rand(300,2)  fig = plt.figure()  ax1 = fig.add\_subplot(2,1,1)  ax2 = fig.add\_subplot(2,1,2)  ax1.scatter(X[:,0],X[:,1], s=0.5, c='black')  h = hopkins\_statistic(X)  ax1.set\_title(f'hopkins统计量：{h:0.2f}')  X1 = np.random.randn(150,2)+2  X2 = np.random.randn(150,2)+5  X = np.concatenate((X1, X2), axis=0)  ax2.scatter(X[:,0],X[:,1], s=0.5, c='black')  h = hopkins\_statistic(X)  ax2.set\_title(f'hopkins统计量：{h:0.2f}')  plt.show() |

这里定义了一个hopkins统计量的计算函数。K

## 7.2 理解聚类结果

聚类算法是无监督学习,它适合用来考察数据本身的特性。即，通过聚类算法来了解获得数据的特性。然而，在聚类算法的实际应用中经常可以发现一些不正确使用聚类算法的地方。例如，市场分析人员拿到用户数据。在进行用户细分时，通过经验认识到用户可以分为四个类。分析人员对用户数据进行聚类，获得四个簇，则认为那每个簇对应一种用户细分。这种想法和做法想当然了。

再举例，有一个客户购买的数据集，每条数据是一个购买者购买一件商品的记录，包括商品的特征。当前数据没有性别特征，有人认为，当前购买者无非是男性或女性。因此将该数据集聚类操作获得两个簇，则两个簇分别描述了男性和女性购买者。这对吗？那如果这个数据集中的商品只有两类奢侈品和日用品（当前数据没有给出该特征），那聚类的结果是反映的则两类商品吗？

到底聚成的两个簇反映的是什么特征不是使用聚类算法的人想当然认为的。对于k-means算法得到的聚类结果，需要考察簇的质心的特征结合应用背景来推断簇所描述的类。

## 7.3 DBSCAN的参数选取

DBSCAN算法中有两个关键参数：邻域半径Eps和密度阈值Minpts。它们的选取对聚类性能有很大的影响。

## 7.4 案例：客户细分

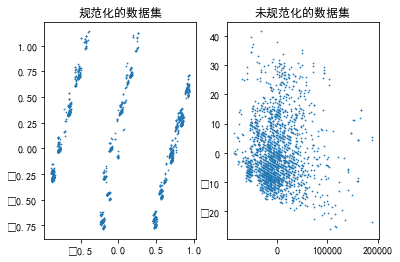
市场细分（market segmentation）或客户细分（Customer Segmentation）是最基本的营销活动。当一家公司不能和所有的潜在客户建立连接，它们必须进行市场划分，即将用户划分成不同的组。有相似的需求的客户在同一个组。然后公司将自己的产品定位到一个细分的市场，即一个用户组。聚类算法是基于数据将市场进行细分的有效方法。

本例子使用kaggle上的一个客户数据集，共2000条客户数据，包含七个属性：

|  |  |
| --- | --- |
| 属性名 | 描述 |
| Sex | 性别：0 男、1 女 |
| Marital status | {0,1} Marital status of a customer.  0 single;  1 non-single (divorced / separated / married / widowed) |
| Age | Integer The age of the customer in years, calculated as current year minus the year of birth of the customer at the time of creation of the dataset  18 Min value (the lowest age observed in the dataset)  76 Max value (the highest age observed in the dataset) |
| Education | {0,1,2,3} Level of education of the customer  0 other / unknown  1 high school  2 university  3 graduate school |
| Income | Real Self-reported annual income in US dollars of the customer. |
| Occupation | {0,1,2} Category of occupation of the customer.  0 unemployed / unskilled  1 skilled employee / official  2 management / self-employed / highly qualified employee / officer |
| Settlement size | {0,1,2} The size of the city that the customer lives in.  0 small city  1 mid-sized city  2 big city |

上面的类别属性，也都是序列属性。例如，occupation属性的三个值{0，1，2}描述了职业水平。因此，这里都把它们当作数值属性。但需要规范化。下面的代码读入数据集，考察缺失值，但没有发现。然后进行最大-最小值规范化。然后将数据集用PCA处理后采用前两个主成分进行可视化。

|  |
| --- |
| df = pd.read\_csv('data/segmentation\_data.csv')  df = df.drop(['ID'], axis=1)  # 1. 考察数据集是否有缺失值  for col in df.columns:  s=pd.isnull(df[col]).sum()  print(f'{col}: {s}')  # 2. max-min规范化  scaler = MinMaxScaler()  # scaler = StandardScaler()  X = scaler.fit\_transform(df)  # 3. PCA可视化  plt.rcParams['font.sans-serif']=['SimHei']  fig = plt.figure()  ax1 = fig.add\_subplot(1,2,1)  ax2 = fig.add\_subplot(1,2,2)  pca = PCA(n\_components = 2)  X\_pca\_norm = pca.fit\_transform(X)  ax1.scatter(X\_pca\_norm[:, 0], X\_pca\_norm[:, 1],s=0.5)  ax1.set\_title('规范化的数据集')  X\_pca\_unnorm = pca.fit\_transform(df.values)  ax2.scatter(X\_pca\_unnorm[:, 0], X\_pca\_unnorm[:, 1],s=0.5)  ax2.set\_title('未规范化的数据集')  plt.show() |



可以看到规范化后的数据集呈现聚集特性。然后计算霍普金斯统计量，考察聚类趋势。

|  |
| --- |
| # 4. 计算hopkins统计量  def hopkins\_statistic(X):  sample\_size = int(X.shape[0]\*0.05)  X\_uniform\_random\_sample = uniform(X.min(axis=0), X.max(axis=0) ,(sample\_size , X.shape[1]))  random\_indices=sample(range(0, X.shape[0], 1), sample\_size)  X\_sample = X[random\_indices]  neigh = NearestNeighbors(n\_neighbors=2)  nbrs=neigh.fit(X)  u\_distances , u\_indices = nbrs.kneighbors(X\_uniform\_random\_sample , n\_neighbors=2)  u\_distances = u\_distances[: , 0]  w\_distances , w\_indices = nbrs.kneighbors(X\_sample , n\_neighbors=2)  w\_distances = w\_distances[: , 1]  u\_sum = np.sum(u\_distances)  w\_sum = np.sum(w\_distances)  H = u\_sum/ (u\_sum + w\_sum)  return H  h = hopkins\_statistic(df.values)  print(f'未规范化数据集的霍普金斯统计量：{h}')  h = hopkins\_statistic(X)  print(f'规范化数据集的霍普金斯统计量：{h}') |

得到结果

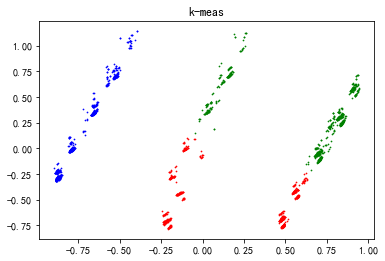
未规范化数据集的霍普金斯统计量：0.9441134762070311

规范化数据集的霍普金斯统计量：0.9630013682219151

可见，无论是在规范化或未规范化的数据集上均表现出了很强的聚类趋势。只不过未规范化的数据集不适合用PCA可视化，因为没有观察到什么结果。

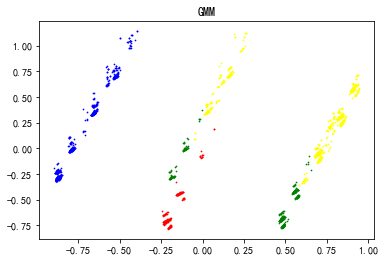
下面用k-means算法聚类并可视化聚类结果。此时是在规划化的数据集，但不是PCA处理后的主成分上进行聚类。

|  |
| --- |
| #%% 5. kmeans  from sklearn.cluster import KMeans  model = KMeans(n\_clusters = 3, random\_state = 12345, n\_init=20)  y\_pred = model.fit\_predict(X)  cluster = set(y\_pred)  for c,u in zip(colors, cluster):  idx = y\_pred==u  plt.scatter(X\_pca\_norm[y\_pred==u,0], X\_pca\_norm[y\_pred==u,1], c=c, s=0.5)  plt.title('k-meas')  plt.show() |



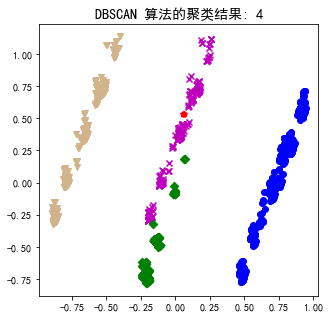
继续用高斯混合模型，进行聚类，并可视化结果。

|  |
| --- |
| #%% 6. GMM  from sklearn.mixture import GaussianMixture  K = 4  model = GaussianMixture(n\_components=K, covariance\_type='full', random\_state=15)  y\_pred = model.fit\_predict(X)  cluster = set(y\_pred)  print(cluster)  for c,u in zip(colors, cluster):  idx = y\_pred==u  plt.scatter(X\_pca\_norm[y\_pred==u,0], X\_pca\_norm[y\_pred==u,1], c=c, s=0.5)  plt.title('GMM')  plt.show() |



进一步使用DBSCAN进行聚类，并可视化结果。

|  |
| --- |
| #%% 7. DBSCAN  from sklearn.cluster import DBSCAN  model = DBSCAN(eps=0.7, min\_samples= 10)  y\_pred = model.fit\_predict(X)  print(f'clusters: {set(y\_pred)}')  n\_clusters = len(set(y\_pred)) - (1 if -1 in y\_pred else 0)  core\_samples\_mask = np.zeros\_like(model.labels\_, dtype=bool)  core\_samples\_mask[model.core\_sample\_indices\_] = True  set\_color=['b','tan','m','g','c','k','r']  set\_marker=['o','v','x','D','>','p','<']  #绘制 DBSCAN 的聚类结果  plt.figure(figsize=(5, 5))  unique\_labels = set(y\_pred)  i = -1 #flag  for k, col in zip(unique\_labels, set\_color[0:len(unique\_labels)]):  if k == -1:  col = 'k' # 黑色表示标记噪声点.  class\_member\_mask = (y\_pred == k)  i += 1  if (i>=len(unique\_labels)): i = 0    # 绘制核心对象  xcore = X\_pca\_norm[class\_member\_mask & core\_samples\_mask]  plt.scatter(xcore[:, 0], xcore[:, 1], marker=set\_marker[i], c=col)  xncore = X\_pca\_norm[class\_member\_mask & ~core\_samples\_mask]    # 绘制边界对象, 红色  plt.scatter(xncore[:, 0], xncore[:, 1], marker='p', c='r')  plt.title('DBSCAN 算法的聚类结果: %d' % n\_clusters,fontsize=14)  plt.show() |



从可视化的效果来看，DBSCAN的聚类效果最好。下面再DBSCAN 的聚类结果上考察每个簇的特性。首先计算每个簇的质心向量。然后，用scaler.inverse\_transform([vec])将质心向量反向规范化回原始的数据尺度。

|  |
| --- |
| #%% 8. 理解聚类结果  for i in cluster:  idx = y\_pred==i  print(f'cluster {i}:')  vec = np.mean(X[idx], axis=0)  centroid = scaler.inverse\_transform([vec])  for item in centroid[0]:  print(f'{item:0.2f}', end=' ')  print() |

下面分结合特征描述，分析每个簇客户的用户群体。

**cluster 0**: 0.00 0.00 39.09 0.79 129262.82 0.93 0.98

男性；单身；39岁左右；教育程度high school偏低；年收入12万左右；职业skilled employee偏上；居住在中等城市。

**cluster 1**: 1.00 1.00 33.19 1.26 116442.17 0.74 0.57

女性；non-single；33岁左右；教育程度high-school偏上；年收入11万左右；职业；skilled employee / official要偏低；居住倾向于中、小城市

**cluster 2**: 0.00 1.00 33.97 1.27 122976.72 0.93 0.92

男性、non-single；33岁左右；教育程度high-school偏上；年收入12万左右；职业倾向于有技巧型的；居住在中等城市

**cluster 3**: 0.00 35.15 0.93 97997.24 0.37 0.07

女性；单身；35岁左右；年收入9万；职业偏向于unemployed / unskilled；居住在小城市

# 第八章：特征工程实验

## 8.1 特征选择

本节首先实验一下特征选择对模型性能的影响。和特征选择的工具。

## 8.2 特征工程工具

Openfe包提供了特征工程工具箱

# 第九章：关联规则分析：零售业的分析工具

## 9.1 最大频繁项集和频繁闭项集

首先介绍教材中未涉及的两个概念最大频繁项集和闭频繁项集。

1.闭频繁项集：当阈值设置的不高时，从数据集中会挖掘出大量的频繁项集。一种对付该问题的方法是挖掘闭频繁项集。一个项集X称作是闭项集，当不存在一个X的超集Y，它和X具有同样的支持度。例如，有一个交易记录集合{I1，I2}，{I1, I2, I3}。项集{I1}就不是闭项集，因为它的超集{I1，I2}和项集{I1}具有相同的支持度。{I1, I2}, {I1, I2, I3}是闭项集。X如果是频繁的，且是闭项集，则称X是闭频繁项集。

2. 最大频繁项集：因为频繁项集的子集也是频繁的，因此通常的任务是挖掘最大频繁项集。最大频繁项集定义为，一个项集它是最大频繁项集只要满足它不是其他频繁项集的子集。上面的例子中，如果设置支持度阈值是50%（要求大于等于50%），则频繁项集有{I1}、{I2}、{I3}、{I1、I2}、{I1、I2、I3}。但只有一个最大频繁项集{I1, I2, I3}。

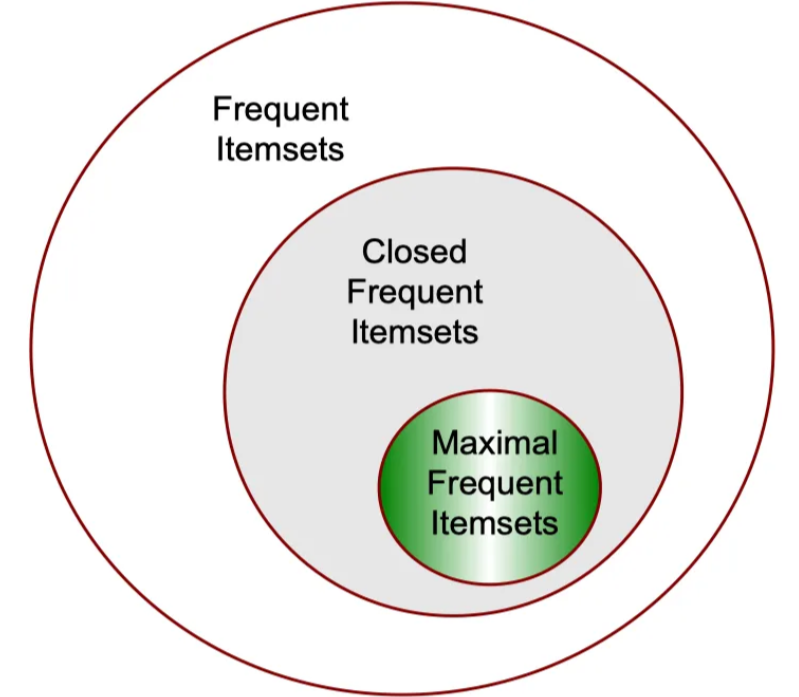


图9.1 频繁项集、闭频繁项集和最大频繁项集的关系

频繁项集、闭频繁项集和最大频繁项集的关系如图所示。闭频繁项集是频繁项集的子集，而最大频繁项集是闭频繁项集的子集。因此，通常的实践中，为了解决挖掘出太多频繁项集的问题，通常是挖掘最大频繁项集。MLXtend库提供了采用fpgrowth挖掘最大频繁项集的函数fpmax

fpmax(df, min\_support=0.5, use\_colnames=False, max\_len=None, verbose=0)

df是使用mlxtend.preprocessing模块下的TransactionEncoder函数预处理了的数据帧。它返回的结果是一个数据帧包含两列 ['support', 'itemsets']

代码9-1：挖掘最大频繁项集

|  |
| --- |
| import pandas as pd  from mlxtend.preprocessing import TransactionEncoder  from mlxtend.frequent\_patterns import fpmax, association\_rules  itemSetList = [['A', 'C', 'D'],  ['B', 'C', 'E'],  ['A', 'B', 'C','E'],  ['B', 'E']]  #数据预处理——编码  te = TransactionEncoder()  te\_array = te.fit(itemSetList).transform(itemSetList)  df = pd.DataFrame(te\_array, columns=te.columns\_)  #利用 FP-Growth 算法生成频繁项集，最小支持度为 0.5  frequent\_itemsets = fpmax(df, min\_support=0.5, use\_colnames=True)  print("发现的频繁项集包括： \n", frequent\_itemsets) |

上面程序运行的结果是：

|  |
| --- |
| 发现的最大频繁项集包括：  support itemsets  0 0.5 (A, C)  1 0.5 (B, C, E) |

在支持度阈值为0.5时，挖掘到了两个最大频繁项集。(A, C)和(B, C, E)

而对比挖掘得到的频繁项集

|  |
| --- |
| 发现的频繁项集包括：  support itemsets  0 0.75 (C)  1 0.50 (A)  2 0.75 (E)  3 0.75 (B)  4 0.50 (C, E)  5 0.50 (A, C)  6 0.75 (B, E)  7 0.50 (B, C)  8 0.50 (B, C, E) |

可以看到最大频繁项集更精简了。

## 9.2 零售业的营销策略

零售业里，老板考核员工的主要指标是商品的销售额。如此每个员工只会关心自己负责的商品的销售情况。卖啤酒的只管卖啤酒，卖尿布的不会操心别人负责的商品。在沃尔玛的卖场管理体系中，购物篮是主要管理对象，而不仅仅是商品。这是因为沃尔玛认为购物篮里透露出的商品关联和用户购买行为合理的利用可以促进超市销售的增长。因此在商场的层面会有部门负责购物篮分析和其后交叉销售等营销策略实施。

沃尔玛根据商品在卖场中的表现，将商品分为几个类别：

（1）主力商品。销售金额靠前的商品，实现了主要的销售数量和销售额。营销学领用的80/20定律是指20%的商品创造了80%的销售额。这20%的商品即主力商品（也有的说是25%）。但不是主力商品数量越多越好。高于或低于这个比例零售业的业绩都会受影响。

（2）辅助商品。从品种的深度、广度对主力商品进行补充的商品。通过这些商品可以增加主力商品的销售额。因为他们吸引了客户的光顾。商店里如果没有辅助商品，商店的陈列就会很单调，客户则失去选择余地，从而失去了购物的乐趣，导致主力商品销售不畅。

购物篮分析可以帮助零售业者建立商品之间的关联特别是分析和主力商品相关联的辅助商品，实施营销策略。

（1）在设计货架时，将经常和主力商品一起购买的商品摆放在一起增加交叉销售（cross selling）的机会

（2）进行促销。一个案例是一家超市将鱼缸降价促销，反而带动了相关的鱼食、加氧气泵等辅助商品的热销。

（3）二次陈列。即交叉陈列，将原本在不同区域的关联度高的商品在专门的区域一起摆放陈列。

Real Link是沃尔玛的一套面向供货商的管理系统。我们看一下沃尔玛的Retail Link系统主要包括的购物篮分析。

（1）商品购物篮数量。不仅向供应商提供商品销售的数量和金额，还提供出现在购物篮的次数。供货商可以据此可以获得更丰富的销售信息。

（2）购物篮表现分析。可以帮助供货商改善商品。一个案例是某个便利店每个周末的早上切片面包卖的非常好。经调查是来到该商店附件的钓鱼爱好者喜欢买切片面包做鱼饵。如此供货商开发了适用于做鱼饵的廉价面包。

（3）客户购买频率分析。根据使用会员卡搜集到的客户购买信息，如一年的购买信息，可以建立周期性的客户购买商品预测模型。

（4）商品关联分析。帮助确定主力、辅助商品之间的关联。帮助超市确定进货、补货商品的信息等。

商品之间的关联可以分为强关联、弱关联和互斥关系。强关联的商品同时出现在购物篮中的概率很大。比如，大家都可以理解豆浆和油条、牛奶和面包都是强关系。弱关联的关系表示两个商品之间的关系并不稳定，通过强诱导才会发生关联。例如在周末才会发生的啤酒和尿布。互斥关系表示两件商品几乎不会同时出现在购物篮中。通常顾客在购买这类商品时目的性很强。如大型家电、家具、药品等。根据商品的关联关系不同超市可以实施不同的营销策略。

**9.2.1 规划商品的陈列**

超市为了促进商品的销售有这样一种策略，将强关系的商品，即最可能会一起购买两件商品摆放在很远的距离，如此顾客从一个商品的位置走到另一个商品的位置途中会浏览更多的商品。而途中摆放促销的商品更会刺激客户的消费。下面的例子将从超市以往的销售数据找出这样一对最有可能一起购买的商品。

本文使用一个文件grocery.csv来做分析。它包含一个超市的一个月的交易数据。有9835条交易记录，169个项目。首先，读入数据集

代码9-2：商品陈列

|  |
| --- |
| import pandas as pd  from mlxtend.preprocessing import TransactionEncoder  from mlxtend.frequent\_patterns import fpgrowth, association\_rules  # 1. 读grocery.csv, 创建数据集tlist  with open('data/grocery.csv') as f:  lines = f.readlines()    tlist = []  for line in lines:  tlist.append(line.split(','))  te = TransactionEncoder()  te\_array = te.fit(tlist).transform(tlist)  df = pd.DataFrame(te\_array, columns=te.columns\_) |

创建的数据帧df就是预处理后的数据集。

当前的任务，发现这样一对最有可能被一起购买的商品，其实就是一个从数据集中发现一个最频繁的二项集。

|  |
| --- |
| fis = fpgrowth(df, min\_support=0.05, max\_len=2, use\_colnames=True)  fis.sort\_values(by=['support'], inplace=True, ascending=False)  for index, row in fis.iterrows():  if len(row['itemsets'])==2:  print(row['support'], row['itemsets']) |

上面的代码挖掘频繁二项集，并按照支持度降序排序。挖掘得到的结果fis是pandas的数据帧结构。分析人员可以一开始把参数min\_support设的小一点，否则有可能不能发现有效的频繁项集。fpgrowth算法提供了参数max\_len可以设置项集的最大长度。因为只是想发现频繁二项集，而算法没有提供项集的最短长度，因此上面的代码遍历fis，只输出频繁二项集。结果如下：

|  |
| --- |
| 0.07269954245043213 frozenset({'whole milk', 'other vegetables'})  0.05327910523640061 frozenset({'whole milk', 'yogurt'}) |

可以看到牛奶和蔬菜是最常一起购买的两种商品。那么还有一个问题，超市管理者应该把两者中的哪件商品放到超市最深的角落呢？答案是最经常被购买的那种商品。下面的代码挖掘频繁一项集，并降序排序。

|  |
| --- |
| #%% 3. 考察 whole milk', 'other vegetables 最频繁项  fis = fpgrowth(df, min\_support=0.05, max\_len=1, use\_colnames=True)  fis.sort\_values(by=['support'], inplace=True, ascending=False)  print(fis) |

输出的结果中可以找到 两个商品

|  |
| --- |
| 4 0.235079 (whole milk)  5 0.180173 (other vegetables) |

明显，牛奶有更高的支持度，因此，在设计摆放商品时，牛奶应该被放在超市最深的角落。而蔬菜到牛奶的路径可以尽量长一些。

**9.4.2 商品促销**

现在有一种商品A要促销，超市管理者希望可以找到一种商品B，它和A的关联度很高，而且B自身的销售情况又很好（B的销售数量很高，希望能带动A的销售）。将A和B捆绑销售。分析人员可以把该问题转换为，发现一个商品B，满足B->A的关联规则，且B的支持度也很高。以Groceries数据集为例，假设为商品canned beer做促销。下面的代码挖掘后件出现了‘canned beer’商品的关联规则。获得前件。

代码9-3：商品促销（接代码9-2）

|  |
| --- |
| candidate = 'canned beer'  fis = fpgrowth(df, min\_support=0.005, max\_len=2, use\_colnames=True)  rules = association\_rules(fis, metric='confidence', min\_threshold=0.005, support\_only=False)  rules= rules[rules['lift']>1]  rules.sort\_values(by=['confidence'], inplace=True, ascending=False)  for idx, item in enumerate(rules['consequents']):  if candidate in list(item):  ante = rules.iloc[idx]['antecedents']  ante = list(ante)  cons = rules.iloc[idx]['consequents']  cons = list(cons)  conf = rules.iloc[idx]['confidence']  print(f'{ante}->{cons} {conf:0.3f}') |

上面的代码首先挖掘频繁项集fis，再获得关联规则rules。然后按照提升度lift>1来挑选规则，并按照置信度的降序排序。挖掘得到的规则rules是数据帧结构，因此遍历排序的规则，并判断待查找的商品，即变量candidate中的商品是否出现在了规则的后件中（rules['consequents']）。前件、后件中的项集是用python的frozenset数据结构保存。因此list(item)是把后件转换成了list结构。最后现实满足条件的规则，结果如下：

|  |
| --- |
| ['shopping bags\n']->['canned beer'] 0.116  ['soda']->['canned beer'] 0.081  ['bottled water']->['canned beer'] 0.064  ['rolls/buns']->['canned beer'] 0.047  ['other vegetables']->['canned beer'] 0.037 |

分析人员试图发现B->A的关联规则，并选出购买频数高的商品B。然而从运行结果中发现，购物袋是和啤酒一起购买次数最多的商品。但实际上正确的逻辑应该是购买了啤酒的客户通常会购买购物袋来装啤酒。而不像B->A，所表述的客户买了袋子会买啤酒。也即关联规则实际上表示了关联关系，但不能正确的表示因果关系。

如果想从上面的四种商品：'soda','bottled water','rolls/buns'和'other vegetables'中挑选一个支持度最高的商品作为促销捆绑上面，使用下面的代码获得每种商品的支持度

|  |
| --- |
| candidates = ['soda','bottled water','rolls/buns','other vegetables']  fis = fpgrowth(df, min\_support=0.01, max\_len=1, use\_colnames=True)  for idx, item in enumerate(fis['itemsets']):  item = list(item)  sup = fis.iloc[idx]['support']  if item[0] in candidates:  print(f'{item} {sup:0.3f}') |

得到结果：

['other vegetables'] 0.180

['rolls/buns'] 0.149

['bottled water'] 0.085

['soda'] 0.116

因此，在实际运作中，超市管理者最直接会选'other vegetables'作为canned beer促销的捆绑商品。