**杭州电子科技大学计算机学院**

**数据挖掘**

**实验1：关联规则挖掘**

时间：2024年11月19日 学号：22270336 姓名：成家康

**注意：**

**1）实验开始前，填写学号和姓名。**

**2）将文件名由“关联规则挖掘”更改为“学号+姓名+关联规则挖掘”。**

**3）作业做完验收之后提交至泛雅平台：上机1**

# 一、实验目的

1、了解关联规则的基本思想，理解频繁项集挖掘的一般过程和基本原理；

2、巩固Apriori算法的算法思想，能够进行关联规则分析。

# 二、实验原理

1、频繁项集

设I={i1, i2,…,im}是项的集合，D是数据库事务的集合，其中每个事务T是一个非空项集，使得T包含于I。每一个事务具有唯一的事务标识符TID。设A是一个项集，事务T包含A，当且仅当A包含于T。如果项集A中包含k个项，则称其为k项集。项集的出现频度是包含项集的事务数，简称为项集的频度、支持度计数或计数。如果项集I的相对支持度满足预定义的最小支持度阈值，则I是频繁项集。

2、关联规则、关联规则的挖掘

关联规则是形如A=>B的逻辑蕴含式，其中A包含于I，B包含于I，A不等于空集，B不等于空集，并且A交B等于空集。如果事务数据库D中有s%的事务包含A并B，则称关联规则X=>Y的支持度为s%，实际上，支持度是一个概率值。若项集A的支持度记为support(A)，规则的置信度为support(A并B)／support (A)。这是一个条件概率P(B|A)。也就是：

support(A=>B)=P(A并B)

confidence(A=>B)=P(B|A)=support(A并B)／support(A)

同时满足最小支持度阈值（min\_sup）和最小置信度阈值（min\_conf）的规则称为强规则。

为方便计算，用0%~100%之间的值，而不是0.0~1.0之间的值来表示支持度和置信度。

confidence(A=>B)式表明，规则A=>B的置信度容易从A并B的支持度计数推出。也就是说，一旦得到A、B和A并B的支持度计数，则导出对应的关联规则A=>B和B=>A，并检查它们是否是强规则是直截了当的。因此，挖掘关联规则的问题可以归结为挖掘频繁项集。

一般而言，关联规则的挖掘是一个两步的过程：

（1）找出所有的频繁项集：根据定义，这些项集的每一个频繁出现的次数至少与预定义的最小支持度计数min\_sup一样。

（2）有频繁项集产生强关联规则：根据定义，这些规则必须满足最小支持度和最小置信度。

由于第二步的开销远低于第一步，因此挖掘关联规则的总体性能由第一步决定。

3、Apriori算法

**（1）Apriori 算法概述**

Apriori算法是一种挖掘关联规则的频繁项集算法，其核心思想是通过候选集生成和情节的向下封闭检测两个阶段来挖掘频繁项集。而且算法已经被广泛的应用到商业、网络安全等各个领域。

**（2）问题的引入**

购物篮分析：引发性例子

Question：哪组商品顾客可能会在一次购物时同时购买？

关联分析

Solutions：

1：经常同时购买的商品可以摆近一点，以便进一步刺激这些商品一起销售。

2：规划哪些附属商品可以降价销售，以便刺激主体商品的捆绑销售。

**（3）Apriori算法**过程分为两个步骤：

* 第一步通过迭代，检索出事务数据库中的所有频繁项集，即支持度不低于用户设定的阈值的项集；
* 第二步利用频繁项集构造出满足用户最小信任度的规则。

具体做法就是：

首先找出频繁1-项集，记为L1；然后利用L1来产生候选项集C2，对C2中的项进行判定挖掘出L2，即频繁2-项集；不断如此循环下去直到无法发现更多的频繁k-项集为止。每挖掘一层Lk就需要扫描整个数据库一遍。算法利用了一个性质：

Apriori 性质：任一频繁项集的所有非空子集也必须是频繁的。意思就是说，生成一个k-itemset的候选项时，如果这个候选项有子集不在(k-1)-itemset(已经确定是frequent的)中时，那么这个候选项就不用拿去和支持度判断了，直接删除。具体而言：

1） 连接步

为找出Lk（所有的频繁k项集的集合），通过将Lk-1（所有的频繁k-1项集的集合）与自身连接产生候选k项集的集合。候选集合记作Ck。设l1和l2是Lk-1中的成员。记li[j]表示li中的第j项。假设Apriori算法对事务或项集中的项按字典次序排序，即对于（k-1）项集li，li[1]<li[2]<……….<li[k-1]。将Lk-1与自身连接，如果(l1[1]=l2[1])&&( l1[2]=l2[2])&&……..&& (l1[k-2]=l2[k-2])&&(l1[k-1]<l2[k-1])，那认为l1和l2是可连接。连接l1和l2 产生的结果是{l1[1],l1[2],……,l1[k-1],l2[k-1]}。

2） 剪枝步

CK是LK的超集，也就是说，CK的成员可能是也可能不是频繁的。通过扫描所有的事务（交易），确定CK中每个候选的计数，判断是否小于最小支持度计数，如果不是，则认为该候选是频繁的。为了压缩Ck,可以利用Apriori性质：任一频繁项集的所有非空子集也必须是频繁的，反之，如果某个候选的非空子集不是频繁的，那么该候选肯定不是频繁的，从而可以将其从CK中删除。

# 三、实验内容

**1） 实验要求：利用Apriori算法求出事务表中的频繁项集和关联规则**

实验数据：10个购物篮事务构成如表1所示事务表，事务表由项目集I={ I1，I2，I3，I4，I5，I6}组成（项目集I中个项目对应的信息为：I1-牛奶，I2-啤酒，I3-尿布，I4-面包，I5-黄油，I6-饼干）。 其中，最小支持度设定为30%，最小置信度设定为70%。

表1.事务表

|  |  |
| --- | --- |
| TID | 商品 |
| T1 | 牛奶，面包，黄油 |
| T2 | 牛奶，啤酒，尿布 |
| T3 | 面包，黄油，饼干 |
| T4 | 牛奶，尿布，饼干 |
| T5 | 啤酒，尿布 |
| T6 | 牛奶，尿布，面包，黄油 |
| T7 | 啤酒，饼干 |
| T8 | 啤酒，尿布，饼干 |
| T9 | 牛奶，尿布，面包，黄油 |
| T10 | 尿布，面包，黄油 |

**2） 基于关联规则的商品推荐**

主要思想：(1) 借助Apriori算法或FP-growth算法生成频繁项集；

(2) 根据频繁项集生成关联规则（参数可自行设定）；

(3) 根据用户喜爱的电影，利用生成的关联规则进行电影推荐；

(4) 数据集：ml-1m.zip

# 四、实验步骤

（1）实验环境

Win11

pycharm

python

（2）步骤说明（文字+代码）

1）购物篮事务（频繁项集挖掘；生成关联规则）

**步骤 1: 数据准备**

数据以字典形式存储，其中键为事务 ID，值为该事务中的商品列表。通过 Pandas DataFrame 转换为布尔型矩阵，方便后续计算。

说明：

使用字典构造事务列表，每个事务包含多个商品。

将事务数据转换为布尔矩阵，行表示事务，列表示商品，值为 True/False，表示某商品是否出现在对应事务中。

import pandas as pd

data = {

'T1': ['牛奶', '面包', '黄油'],

'T2': ['牛奶', '啤酒', '尿布'],

'T3': ['面包', '黄油', '饼干'],

'T4': ['牛奶', '尿布', '饼干'],

'T5': ['啤酒', '尿布'],

'T6': ['牛奶', '尿布', '面包', '黄油'],

'T7': ['啤酒', '饼干'],

'T8': ['啤酒', '尿布', '饼干'],

'T9': ['牛奶', '尿布', '面包', '黄油'],

'T10': ['尿布', '面包', '黄油']

}

transactions = pd.DataFrame(

[{item: (item in data[tid]) for item in ['牛奶', '啤酒', '尿布', '面包', '黄油', '饼干']} for tid in data])

**步骤 2: 计算支持度**

说明：

itemset 是商品项集。

使用布尔矩阵筛选事务，计算所有事务中包含该项集的比例作为支持度。

def calculate\_support(itemset, transactions):

count = sum(transactions[list(itemset)].all(axis=1))

support = count / len(transactions)

return support

**步骤 3: 实现 Apriori 算法**

说明：

1.初始化单项集：

将每个商品单独作为一个 frozenset。

2.支持度计算与筛选：

对每个项集计算支持度，保留支持度大于等于 min\_support 的项集。

3.生成候选项集：

合并当前项集，生成更大尺寸的候选项集。

4.终止条件：

当前项集为空时停止。

def apriori(transactions, min\_support):

itemsets = []

support\_data = {}

single\_items = [frozenset([item]) for item in transactions.columns]

current\_itemsets = single\_items

while current\_itemsets:

# 计算支持度

current\_supports = {item: calculate\_support(item, transactions) for item in current\_itemsets}

# 筛选满足支持度的项集

current\_itemsets = [item for item, support in current\_supports.items() if support >= min\_support]

support\_data.update(current\_supports)

itemsets.extend(current\_itemsets)

# 生成候选项集

if current\_itemsets:

merged\_set = set().union(\*[set(item) for item in current\_itemsets])

current\_itemsets = list(map(frozenset, combinations(merged\_set, len(current\_itemsets[0]) + 1)))

else:

current\_itemsets = []

return itemsets, support\_data

**步骤 4: 生成关联规则**

说明：

遍历频繁项集，构造可能的前件（antecedent）和后件（consequence）。

根据支持度计算置信度，筛选置信度大于等于 min\_confidence 的规则。

def generate\_rules(frequent\_itemsets, support\_data, min\_confidence):

rules = []

for itemset in frequent\_itemsets:

if len(itemset) < 2:

continue

for consequence in map(frozenset, combinations(itemset, 1)):

antecedent = itemset - consequence

if antecedent:

confidence = support\_data[itemset] / support\_data[antecedent]

if confidence >= min\_confidence:

rules.append((antecedent, consequence, confidence))

return rules

**步骤 5: 执行算法并打印结果**

min\_support = 0.3

min\_confidence = 0.7

frequent\_itemsets, support\_data = apriori(transactions, min\_support)

# 打印频繁项集

print("频繁项集:")

for itemset in frequent\_itemsets:

print(f"{set(itemset)}: {support\_data[itemset]:.2f}")

# 生成关联规则

rules = generate\_rules(frequent\_itemsets, support\_data, min\_confidence)

# 打印关联规则

print("\n关联规则:")

for antecedent, consequence, confidence in rules:

print(f"{set(antecedent)} => {set(consequence)}, 置信度: {confidence:.2f}")

2）电影推荐（数据预处理；频繁项集挖掘；根据频繁项集生成关联规则；根据用户喜爱的电影，利用生成的关联规则进行电影推荐）

**步骤1：导入必要的库和设置**

import pandas as pdfrom mlxtend.frequent\_patterns

import apriori, association\_rulesfrom scipy.sparse

import csr\_matrix

pandas：用于数据读取和处理。

mlxtend：提供 apriori 和 association\_rules 函数，用于频繁项集挖掘和关联规则分析。

scipy：用于将用户电影评分矩阵转化为稀疏矩阵。

pd.set\_option('display.max\_columns', None)

pd.set\_option('display.expand\_frame\_repr', False)

pd.set\_option('display.max\_colwidth', None)

设置显示选项，方便查看完整的 DataFrame 内容。

**步骤2：读取数据**

movies = pd.read\_csv('movies.dat', sep='::', engine='python', encoding='latin1',

names=['MovieID', 'Title', 'Genres'])

ratings = pd.read\_csv('ratings.dat', sep='::', engine='python', encoding='latin1',

names=['UserID', 'MovieID', 'Rating', 'Timestamp'])

users = pd.read\_csv('users.dat', sep='::', engine='python', encoding='latin1',

names=['UserID', 'Gender', 'Age', 'Occupation', 'Zip-code'])

读取 movies.dat、ratings.dat 和 users.dat 文件，分别包含电影信息、用户评分和用户信息。

**步骤3：构建用户-电影矩阵**

user\_movie\_matrix = ratings.pivot(index='UserID', columns='MovieID', values='Rating')

user\_movie\_matrix = user\_movie\_matrix.notnull().astype('int')

pivot 将用户（UserID）与电影（MovieID）的评分数据转换为矩阵形式：行：用户。列：电影。值：电影是否被用户评分（1 表示已评分，0 表示未评分）。

**步骤4：转换为稀疏矩阵**

user\_movie\_sparse = csr\_matrix(user\_movie\_matrix)

将用户-电影矩阵转换为稀疏矩阵形式，优化存储和计算效率。

**步骤5：挖掘频繁项集**

frequent\_itemsets = apriori(user\_movie\_matrix, min\_support=0.3, use\_colnames=True)

print("频繁项集：")

print(frequent\_itemsets)

使用 Apriori 算法挖掘频繁项集：

min\_support=0.3：只保留在至少 30% 用户中出现的电影组合。

use\_colnames=True：使用电影编号作为项集名称。

**步骤6：生成关联规则**

rules = association\_rules(frequent\_itemsets, metric="confidence", min\_threshold=0.7)

print("\n关联规则：")

print(rules)

生成关联规则：

metric="confidence"：以置信度作为衡量指标。

min\_threshold=0.7：规则的置信度需达到 70% 以上。

**步骤7：推荐电影函数**

def recommend\_movies(user\_id, rules, top\_n=5):

user\_rated\_movies = set(user\_movie\_matrix.loc[user\_id][user\_movie\_matrix.loc[user\_id] > 0].index)

applicable\_rules = rules[rules['antecedents'].apply(lambda x: len(x & user\_rated\_movies) > 0)]

recommendations = set()

for \_, rule in applicable\_rules.iterrows():

recommendations |= rule['consequents']

recommendations -= user\_rated\_movies

return movies[movies['MovieID'].isin(recommendations)].head(top\_n)

输入参数：

user\_id：需要推荐电影的用户 ID。

rules：关联规则。

top\_n：推荐的电影数量。

步骤：

获取用户已评分电影。

筛选适用于用户的关联规则（前项中含有用户已评分电影）。

提取规则的后项（推荐电影）。

去除用户已评分电影，返回推荐电影。

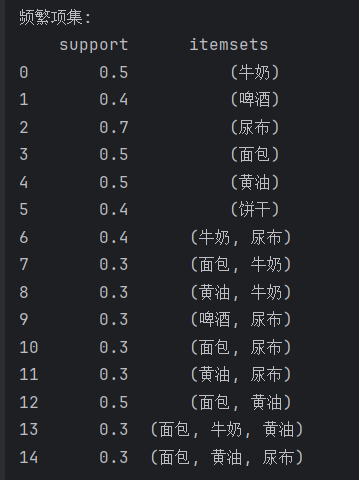
**步骤8：为用户 1 推荐电影**

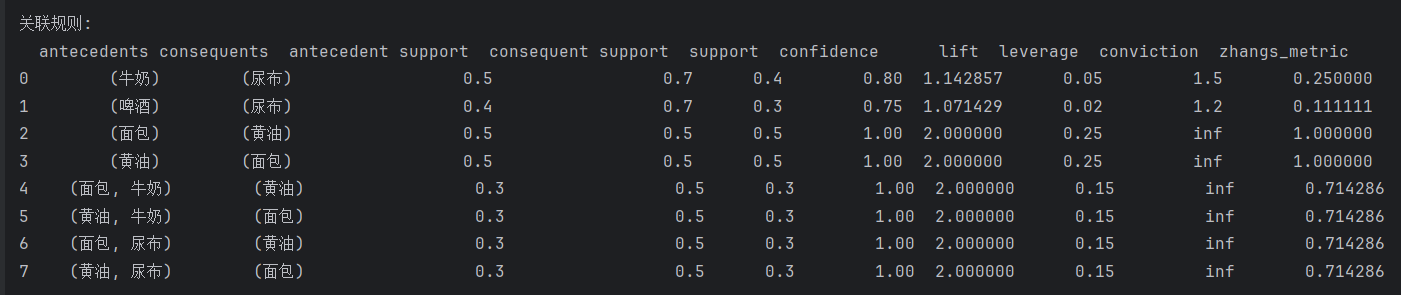
print("\n为用户1推荐的电影：")print(recommend\_movies(1, rules))

基于用户 1 的评分历史和挖掘出的关联规则，为其推荐 5 部未看过的电影。

# 五、实验结果与分析

1. 购物篮事务





1. **频繁项集分析**

根据实验数据，最小支持度设为30%（3次出现），筛选出的频繁项集包括1项、2项和3项频繁项集。以下是详细分析：

1项频繁项集：

支持度最高的是尿布（支持度=0.7），显示尿布是最受欢迎的商品。

其他支持度较高的单项包括牛奶、面包、黄油，表明这些商品也经常出现在事务中。

2项频繁项集：

频繁出现的二元组合包括牛奶和尿布、面包和黄油，分别具有支持度0.4和0.5。

面包和黄油支持度最高，可能表示两者在购物篮中具有较强的关联性。

3项频繁项集：

面包, 牛奶, 黄油和面包, 黄油, 尿布达到了最低支持度0.3，表明这些组合在购物篮中也有一定关联性。

1. **关联规则分析**

关联规则的生成基于最小置信度70%。以下是对部分规则的详细分析：

规则1: (牛奶) → (尿布)

置信度=80%，Lift=1.14。

解释：购物篮中含牛奶时，包含尿布的可能性较大（80%）。但Lift值略高于1，说明关联性较弱。

规则2: (啤酒) → (尿布)

置信度=75%，Lift=1.07。

解释：购买啤酒时，尿布也会有一定概率出现在购物篮中（75%）。但Lift值较低，显示其增强作用有限。

规则3: (面包) → (黄油) 和 (黄油) → (面包)

置信度=100%，Lift=2.0。

解释：面包和黄油之间具有非常强的关联性。Lift值为2，说明在实际中它们共同出现的概率是随机情况的两倍，具有强烈的互补性。

规则4: (面包, 牛奶) → (黄油)

置信度=100%，Lift=2.0。

解释：当购物篮中出现面包和牛奶时，黄油必定会出现在购物篮中。这表明这三个商品可能存在较强的联合购买趋势。

规则5: (黄油, 尿布) → (面包)

置信度=100%，Lift=2.0。

解释：当购物篮中有黄油和尿布时，面包也总是出现。这可能反映了某些特定购物习惯。

**3. 重要性指标解释**

支持度（Support）：项集在所有事务中出现的比例，反映了项集的受欢迎程度。

置信度（Confidence）：规则发生的条件下，结果出现的概率，衡量了规则的可靠性。

提升度（Lift）：实际联合概率与独立概率的比值，Lift > 1表明规则有正相关性。

张氏指标（Zhang's Metric）：用于衡量规则之间的非对称性程度。值越高，规则越重要。

2. 电影推荐

频繁项集：

support itemsets

0 0.343874 (1)

1 0.404470 (110)

2 0.495199 (260)

3 0.359437 (296)

4 0.368709 (318)

5 0.363245 (356)

6 0.330298 (457)

7 0.442384 (480)

8 0.381457 (527)

9 0.438576 (589)

10 0.426821 (593)

11 0.416060 (608)

12 0.368046 (858)

13 0.375662 (1097)

14 0.495033 (1196)

15 0.383775 (1197)

16 0.416225 (1198)

17 0.301325 (1200)

18 0.477318 (1210)

19 0.335099 (1214)

20 0.347351 (1240)

21 0.377152 (1265)

22 0.427649 (1270)

23 0.420199 (1580)

24 0.378808 (1617)

25 0.439238 (2028)

26 0.392219 (2396)

27 0.428808 (2571)

28 0.372517 (2628)

29 0.361093 (2716)

30 0.407119 (2762)

31 0.567550 (2858)

32 0.330464 (2916)

33 0.371026 (2997)

34 0.318543 (3578)

35 0.301987 (480, 260)

36 0.312748 (260, 589)

37 0.389901 (1196, 260)

38 0.326490 (260, 1198)

39 0.349834 (1210, 260)

40 0.305464 (260, 1270)

41 0.315894 (2571, 260)

42 0.316060 (480, 589)

43 0.300331 (480, 1196)

44 0.321026 (480, 1580)

45 0.303642 (480, 2571)

46 0.316391 (1196, 589)

47 0.307119 (1580, 589)

48 0.327483 (2571, 589)

49 0.305795 (593, 2858)

50 0.307285 (608, 2858)

51 0.330960 (1196, 1198)

52 0.368874 (1210, 1196)

53 0.315397 (1196, 1270)

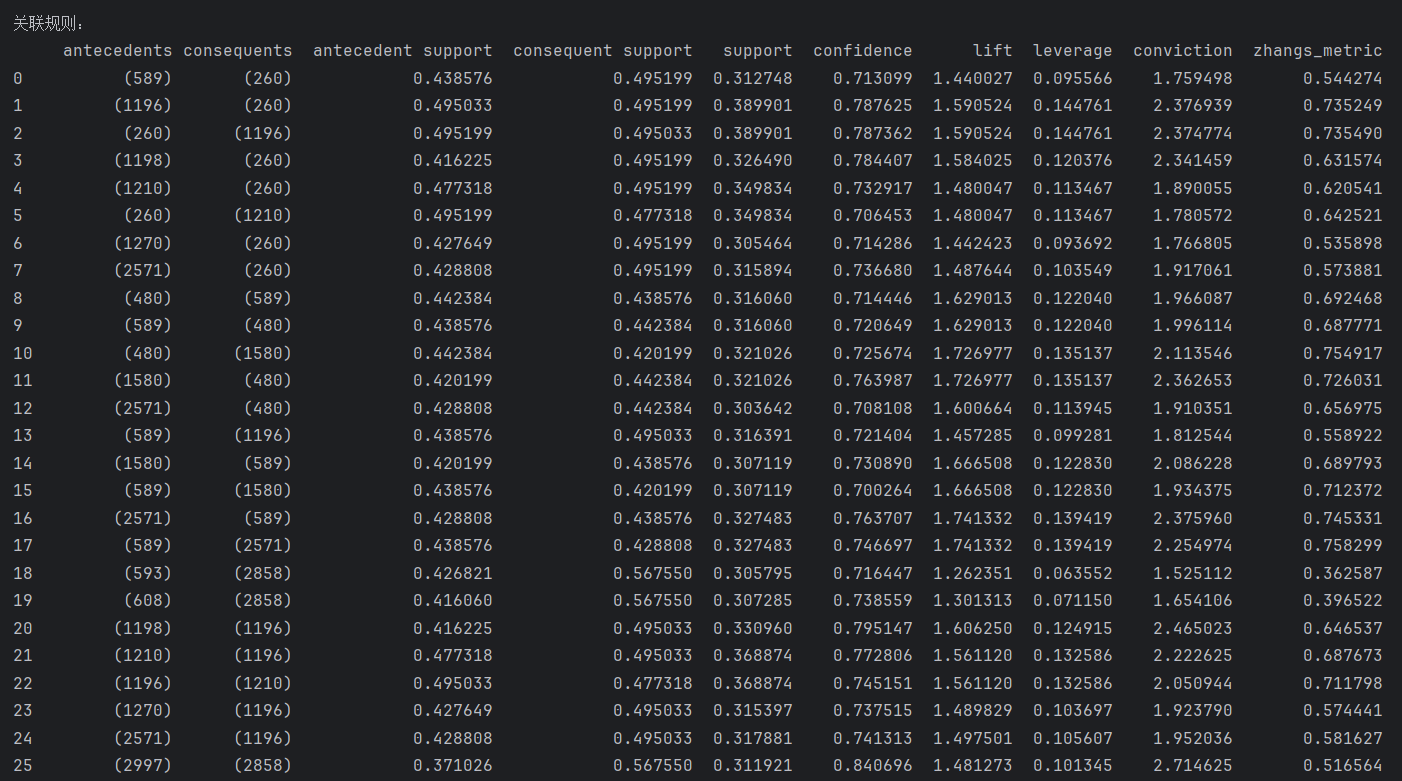
54 0.317881 (2571, 1196)

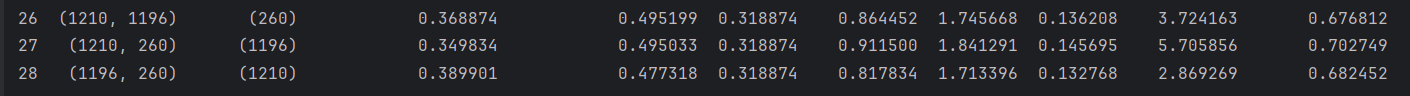
55 0.302815 (2858, 1196)

56 0.311921 (2858, 2997)

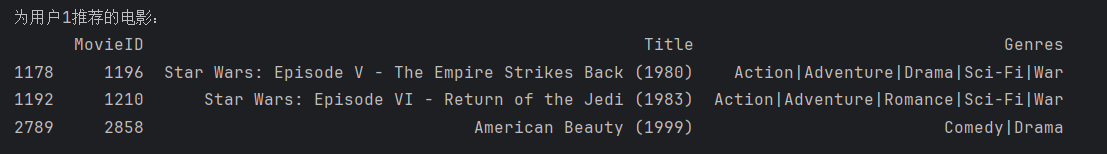
57 0.318874 (1210, 1196, 260)

关联规则：





为用户1推荐的电影：



**1. 频繁项集分析**

从频繁项集结果中可以看出，单项集（如 (1)、(110)、(260)）和多项集（如 (480, 260)、(1210, 1196, 260)）的支持度分布相对较广。这些项集表示在用户评分记录中共同出现的电影组合。例如：

单项集支持度最高的是 (2858)，支持度为 0.567550。这表明电影 American Beauty (1999) 是用户观看率较高的电影。

支持度最高的二项集为 (1210, 1196)，支持度为 0.368874。这意味着很多用户同时观看了 Star Wars: Episode V 和 Star Wars: Episode VI。

**2. 关联规则分析**

关联规则从频繁项集中进一步挖掘了用户行为的关联性，重点关注置信度和提升度：

规则 (1196) -> (260)：置信度 0.7876，提升度 1.5905。这表示用户观看了 Star Wars: Episode V 后，很可能也会观看 Star Wars: Episode IV。

规则 (1210, 1196) -> (260)：置信度 0.8645，提升度 1.7457。这进一步说明，若用户观看了 Star Wars 系列中的两部电影，则观看另一部的可能性更高。

提升度高于 1 表明这些规则在数据集中具有一定的相关性。

**3. 推荐结果分析**

为用户1推荐的电影如下：

Star Wars: Episode V - The Empire Strikes Back (1980)

Star Wars: Episode VI - Return of the Jedi (1983)

American Beauty (1999)

这些推荐基于用户与其他电影的关联关系，结合频繁项集和关联规则分析得出。由于用户1尚未评分这些电影，因此是适合推荐的候选项。

# 实验总结

本次实验通过对关联规则挖掘的实践，深入了解了关联规则的基本思想、频繁项集的挖掘过程以及关联规则分析的具体应用，主要体会和收获如下：

1. 理论理解与实践结合

在实验过程中，我们通过Apriori算法实现了频繁项集的挖掘，熟悉了其两个核心步骤：候选项集的生成与剪枝过程。借助于事务表数据，生成的频繁项集和关联规则很好地展示了商品之间的关联性。尤其是通过支持度和置信度的设定，有效筛选出具有实际意义的强关联规则。

2. Apriori算法的优势与局限

Apriori算法的核心在于利用频繁项集的性质，通过剪枝减少了候选集的数量。然而，算法仍然需要多次扫描事务数据库，当事务量较大时计算成本较高。因此，本次实验也体现了在小规模数据集上，Apriori算法的有效性和实用性，但对于更大规模的数据，可能需要优化或者选择其他算法（如FP-growth）。

3. 商品推荐的实际应用

在关联规则挖掘的基础上，实验进一步将分析结果应用于商品推荐。这一环节展示了如何将数据挖掘的结果用于提升用户体验，如根据用户的购买历史生成个性化推荐。这种方法可以被广泛应用于电商平台，通过精准推荐提高销售转化率。

通过本次实验，不仅加深了对关联规则挖掘的理解，还学习了如何将其应用于实际问题。数据挖掘技术的价值在于将隐藏于海量数据中的模式转化为可操作的信息，为商业决策提供依据。这一过程从理论到实践，从算法到应用，充分体现了数据挖掘技术在现代信息社会中的重要性。

# 七、源代码

import pandas as pd  
from itertools import combinations  
  
# 数据准备  
data = {  
 'T1': ['牛奶', '面包', '黄油'],  
 'T2': ['牛奶', '啤酒', '尿布'],  
 'T3': ['面包', '黄油', '饼干'],  
 'T4': ['牛奶', '尿布', '饼干'],  
 'T5': ['啤酒', '尿布'],  
 'T6': ['牛奶', '尿布', '面包', '黄油'],  
 'T7': ['啤酒', '饼干'],  
 'T8': ['啤酒', '尿布', '饼干'],  
 'T9': ['牛奶', '尿布', '面包', '黄油'],  
 'T10': ['尿布', '面包', '黄油']  
}  
  
transactions = pd.DataFrame(  
 [{item: (item in data[tid]) for item in ['牛奶', '啤酒', '尿布', '面包', '黄油', '饼干']} for tid in data])  
  
  
# 计算支持度  
def calculate\_support(itemset, transactions):  
 count = sum(transactions[list(itemset)].all(axis=1))  
 support = count / len(transactions)  
 return support  
  
  
# 修复 Apriori 算法中生成候选项集的部分  
def apriori(transactions, min\_support):  
 itemsets = []  
 support\_data = {}  
 single\_items = [frozenset([item]) for item in transactions.columns]  
 current\_itemsets = single\_items  
  
 while current\_itemsets:  
 # 计算支持度  
 current\_supports = {item: calculate\_support(item, transactions) for item in current\_itemsets}  
 # 筛选满足支持度的项集  
 current\_itemsets = [item for item, support in current\_supports.items() if support >= min\_support]  
 support\_data.update(current\_supports)  
 itemsets.extend(current\_itemsets)  
 # 生成候选项集  
 if current\_itemsets:  
 # 将 frozenset 转换为 set  
 merged\_set = set().union(\*[set(item) for item in current\_itemsets])  
 current\_itemsets = list(map(frozenset, combinations(merged\_set, len(current\_itemsets[0]) + 1)))  
 else:  
 current\_itemsets = []  
  
 return itemsets, support\_data  
  
  
# 生成关联规则  
def generate\_rules(frequent\_itemsets, support\_data, min\_confidence):  
 rules = []  
 for itemset in frequent\_itemsets:  
 if len(itemset) < 2:  
 continue  
 for consequence in map(frozenset, combinations(itemset, 1)):  
 antecedent = itemset - consequence  
 if antecedent:  
 confidence = support\_data[itemset] / support\_data[antecedent]  
 if confidence >= min\_confidence:  
 rules.append((antecedent, consequence, confidence))  
 return rules  
  
  
# 执行 Apriori 算法  
min\_support = 0.3  
min\_confidence = 0.7  
frequent\_itemsets, support\_data = apriori(transactions, min\_support)  
  
# 打印频繁项集  
print("频繁项集:")  
for itemset in frequent\_itemsets:  
 print(f"{set(itemset)}: {support\_data[itemset]:.2f}")  
  
# 生成关联规则  
rules = generate\_rules(frequent\_itemsets, support\_data, min\_confidence)  
  
# 打印关联规则  
print("\n关联规则:")  
for antecedent, consequence, confidence in rules:  
 print(f"{set(antecedent)} => {set(consequence)}, 置信度: {confidence:.2f}")

import pandas as pd  
from mlxtend.frequent\_patterns import apriori, association\_rules  
from scipy.sparse import csr\_matrix  
  
pd.set\_option('display.max\_columns', None)  
pd.set\_option('display.expand\_frame\_repr', False)  
pd.set\_option('display.max\_colwidth', None)  
  
movies = pd.read\_csv('movies.dat', sep='::', engine='python', encoding='latin1',  
 names=['MovieID', 'Title', 'Genres'])  
ratings = pd.read\_csv('ratings.dat', sep='::', engine='python', encoding='latin1',  
 names=['UserID', 'MovieID', 'Rating', 'Timestamp'])  
users = pd.read\_csv('users.dat', sep='::', engine='python', encoding='latin1',  
 names=['UserID', 'Gender', 'Age', 'Occupation', 'Zip-code'])  
  
user\_movie\_matrix = ratings.pivot(index='UserID', columns='MovieID', values='Rating')  
user\_movie\_matrix = user\_movie\_matrix.notnull().astype('int')  
user\_movie\_sparse = csr\_matrix(user\_movie\_matrix)  
  
frequent\_itemsets = apriori(user\_movie\_matrix, min\_support=0.3, use\_colnames=True)  
print("频繁项集：")  
print(frequent\_itemsets)  
  
rules = association\_rules(frequent\_itemsets, metric="confidence", min\_threshold=0.7)  
print("\n关联规则：")  
print(rules)  
  
def recommend\_movies(user\_id, rules, top\_n=5):  
 user\_rated\_movies = set(user\_movie\_matrix.loc[user\_id][user\_movie\_matrix.loc[user\_id] > 0].index)  
 applicable\_rules = rules[rules['antecedents'].apply(lambda x: len(x & user\_rated\_movies) > 0)]  
 recommendations = set()  
 for \_, rule in applicable\_rules.iterrows():  
 recommendations |= rule['consequents']  
 recommendations -= user\_rated\_movies  
 return movies[movies['MovieID'].isin(recommendations)].head(top\_n)  
  
print("\n为用户1推荐的电影：")  
print(recommend\_movies(1, rules))