第七章

1.实践案例一

import pandas as pd

from mlxtend.frequent\_patterns import apriori #挖掘频繁项集

from mlxtend.frequent\_patterns import association\_rules #挖掘关联规则dataset = {'ID':[1, 2, 3, 4, 5, 6],

'Onion':[1, 0, 0, 1, 1, 1],

'Potato':[1, 1, 0, 1, 1, 1],

'Burger':[1, 1, 0, 0, 1, 1],

'Milk':[0, 1, 1, 1, 0, 1],

'Beer':[0, 0, 1, 0, 1, 0]}

df = pd.DataFrame(dataset)

df = df[['ID', 'Onion', 'Potato', 'Burger', 'Milk', 'Beer']]

df# 利用mlxtend提供的apriori算法函数得到频繁项集，其中设置最小支持度为50%

frequent\_itemsets = apriori(df[['Onion', 'Potato', 'Burger', 'Milk', 'Beer']], min\_support=0.50, use\_colnames=True)

frequent\_itemsetsrules = association\_rules(frequent\_itemsets, num\_itemsets=None, metric='lift', min\_threshold=1) #计算规则，并设置提升度阈值为 1

rules = rules.sort\_values(by="lift", ascending=False) # 按照提升度从大到小进行排序

rules

# 选取提升度大于1.125且置信度大于0.8的关联规则

rules [(rules['lift'] > 1.125) & (rules['confidence'] > 0.8)]

2.实践案例二

import pandas as pd

import numpy as np

import mlxtend

from mlxtend.preprocessing import TransactionEncoder

from mlxtend.frequent\_patterns import fpgrowth

import pyfpgrowthtransaction = [['I1','I2','I5'],

['I2','I4'],

['I2','I3'],

['I1','I2','I4'],

['I1','I3'],

['I2','I3'],

['I1','I3'],

['I1','I2','I3','I5'],

['I1','I2','I3']]

transactionte = TransactionEncoder()

dataset\_te = te.fit(transaction).transform(transaction)

patterns = pyfpgrowth.find\_frequent\_patterns(transaction,2)

patternsrules = pyfpgrowth.generate\_association\_rules(patterns,0.7)

rules

3.实验题目一

import pandas as pd

from mlxtend.frequent\_patterns import apriori #挖掘频繁项集

from mlxtend.frequent\_patterns import association\_rules #挖掘关联规则movies = pd.read\_csv('D:/桌面/movies.csv')

movies.head(10)

# 查看genres类型列的数据

movies['genres']

# 将genres进行one-hot编码，drop('genres', 1) 先把genres列去掉，后面分割之后再拼接上，把genres转换为字符串类型，然后get\_dummies(sep='|') 按竖线进行分割

movies\_hot\_encoded = movies.drop(labels='genres', axis=1).join(movies.genres.str.get\_dummies(sep='|'))

movies\_hot\_encoded

pd.options.display.max\_columns = 100

movies\_hot\_encoded.head(10)

# 将movieId, title同时设置为index(此操作可以把默认生成的第一列index0,1,2,3,...替换掉)

movies\_hot\_encoded.set\_index(['movieId', 'title'], inplace=True)

movies\_hot\_encoded.head()# 挖掘频繁项集，最小支持度为0.02

itemsets = apriori(movies\_hot\_encoded, use\_colnames=True, min\_support=0.02)

itemsets = itemsets.sort\_values(by="support", ascending=False) # 按照支持度从大到小进行输出

print('-' \* 20, '频繁项集', '-' \* 20)

itemsets# 根据频繁项集计算关联规则，设置最小提升度为2

rules = association\_rules(itemsets, num\_itemsets=None, metric='lift', min\_threshold=2)

# 按照提升度从大到小进行排序

rules = rules.sort\_values(by="lift", ascending=False)

print('-' \* 20, '关联规则', '-' \* 20)

rules

4.实验2

import pandas as pd

dataset=pd.read\_csv("D:/桌面/Market\_Basket\_Optimisation.csv")

print(dataset.shape)

dataset.head()# 转换Numpy数组类型

import numpy as np

transaction = []

for i in range(0, dataset.shape[0]):

for j in range(0, dataset.shape[1]):

transaction.append(dataset.values[i,j])

transaction = np.array(transaction)

print(transaction)

# 统计并可视化商品购买频次

import pandas as pd

df = pd.DataFrame(transaction, columns=["items"])

df["incident\_count"] = 1

indexNames = df[df['items'] == "nan" ].index

df.drop(indexNames , inplace=True)

# 对商品名称进行分组，计算每种商品的购买总次数，取前5序号

df\_table =df.groupby("items").sum().sort\_values("incident\_count", ascending=False).reset\_index()

df\_table.head(5).style.background\_gradient(cmap='Greens')# 将每笔交易转换为单独的列表，并将它们收集到Numpy数组中

import numpy as np

transaction = []

for i in range(dataset.shape[0]):

transaction.append([str(dataset.values[i,j]) for j in range(dataset.shape[1])])

transaction = np.array(transaction)

from mlxtend.preprocessing import TransactionEncoder

# 初始化TransactionEncoder并将数据转换为布尔值

te = TransactionEncoder()

te\_ary = te.fit(transaction).transform(transaction)

dataset = pd.DataFrame(te\_ary, columns=te.columns\_)

dataset.head()first30 = df\_table["items"].head(30).values

dataset = dataset.loc[:,first30]

print(dataset.shape)

# 运行FP-growth算法

from mlxtend.frequent\_patterns import fpgrowth

res = fpgrowth(dataset, min\_support=0.05, use\_colnames=True)

res.head(10)from mlxtend.frequent\_patterns import association\_rules

res = association\_rules(res, num\_itemsets=None, metric="lift", min\_threshold=1)

# 根据置信度对值进行排序

res.sort\_values("confidence", ascending=False)

5.实验3

①用Apriori算法进行关联分析

import pandas as pd

import numpy as np

from mlxtend.preprocessing import TransactionEncoder #编码

import timewith open('D:/桌面/basket.txt') as fp:

products = [line.strip().split('\t') for line in fp]

onehot\_enc = TransactionEncoder()

tmp\_array = onehot\_enc.fit\_transform(products)

products\_ = pd.DataFrame(tmp\_array,columns=onehot\_enc.columns\_)

products\_.head(5)t1 = time.time()

from mlxtend.frequent\_patterns import apriori #挖掘频繁项集

# 利用mlxtend提供的apriori算法函数得到频繁项集，其中设置最小支持度为0.1

freq\_items = apriori(products\_, min\_support=0.1,

use\_colnames=True, max\_len=None)

#计算项集的长度

freq\_items['length'] = freq\_items['itemsets'].apply(lambda x: len(x))

freq\_itemsfrom mlxtend.frequent\_patterns import association\_rules #挖掘关联规则

rules = association\_rules(freq\_items,num\_itemsets=None, metric='lift', min\_threshold=1)

# 选取提升度大于1且置信度大于0.5的关联规则

rules[(rules['lift'] > 1) & (rules['confidence'] > 0.5)]

rules = rules.sort\_values(by="lift", ascending=False) # 按照提升度从大到小进行排序

rulesprint(time.time() - t1)

②用FP-Growth算法进行关联分析

import pandas as pd

from mlxtend.preprocessing import TransactionEncoder

from mlxtend.frequent\_patterns import fpgrowth

from mlxtend.frequent\_patterns import association\_rules

import pyfpgrowth

import timewith open('D:/桌面/basket.txt') as fp:

products = [line.strip().split('\t') for line in fp]

onehot\_enc = TransactionEncoder()

tmp\_array = onehot\_enc.fit\_transform(products)

products\_=pd.DataFrame(tmp\_array,columns=onehot\_enc.columns\_)

products\_.head(5)t1 = time.time()

frequent\_itemsets = fpgrowth(products\_, min\_support=0.1, use\_colnames=True)

frequent\_itemsetsrules=association\_rules(frequent\_itemsets, num\_itemsets=None, metric="lift", min\_threshold=1)

rules[(rules['lift'] > 1) & (rules['confidence'] > 0.5)]

rules = rules.sort\_values(by="lift", ascending=False)

rulesprint(time.time() - t1)