Frequent itemset mining——Apriori algorithm

--10215501435 杨茜雅

实验要求：

基于 Apriori 算法完成 Market Basket 分析实验

数据视图：

Variables	Descriptions
InvoiceNo	发票编号（如果此代码以C 开头，则表示操作已取消）
StockCode	产品代码（每个产品的唯一编号）
Description	产品名称
Quantity	产品数量（发票上的产品数量已售出多少）
InvoiceDate	发票日期
Price	统一价格
CustomerID	唯一的客户编号
Country	国家

第一部分：Import Data & Data Preprocessing

●导入并查看数据信息，统计每个属性缺失值数量，处理缺失值（丢弃）；确定已取消的交易并删除；

●异常值处理；（将异常值定义为位于 1% 和 99% 分位数之外的值，并使用阈值来代替数据中的异常值）

●因为每个 stock code 代表一种产品，所以 Description 和 StockCode 的唯一值应该相等，删除代表多种产品 stock codes；

●stock code 中的 POST 表示邮费，并不代表产品，将其删除；

第二部分：Preparing Invoice-Product Matrix for ARL Data Structure

将原始数据转换为适合进行关联规则分析的格式，例如：其中每一行代表一笔交易，每一列代表一个产品，单元格的值表示该产品在该笔交易中是否存在（1 表示存在，0 表示不存在）。

第三部分：Determination of Association Rules

●使用 Apriori 计算 support values，min_support 设置为 0.01；

●从频繁项集中生成关联规则，评估关联规则的指标为 support，最小支持度阈值为 0.01；

●查看支持度最高的前五个关联规则。

第一部分：Import Data & Data Preprocessing

实验数据 retail.xlsx 有两个 work sheet，分别载入 df_1 和 df_2，所有数据处理操作都分别进行两次，以下代码示例已 df_1 操作为例，df_2 类似。

导入数据

```
# 1. 导入数据
df_1 = pd.read_excel('retail.xlsx', sheet_name='Year 2009-2010')  # 替换为实际的工作表名称
df_2 = pd.read_excel('retail.xlsx', sheet_name='Year 2010-2011')  # 替换为实际的工作表名称
# Print the first few rows of the dataframe
print("Year 2009-2010 dataframe:\n")
print(df_1.head())
print("Number of instances (rows):", df_1.shape[0])
print("Number of attributes (columns):", df_1.shape[1])
print("Column names:", df_1.columns.tolist())
print()
```

数据概况：

```
Year 2009-2010 dataframe:

  Invoice StockCode                          Description  Quantity  \
0  489434     85048  15CM CHRISTMAS GLASS BALL 20 LIGHTS        12
1  489434    79323P                   PINK CHERRY LIGHTS        12
2  489434    79323W                  WHITE CHERRY LIGHTS        12
3  489434     22041         RECORD FRAME 7" SINGLE SIZE         48
4  489434     21232       STRAWBERRY CERAMIC TRINKET BOX        24

          InvoiceDate  Price  Customer ID         Country
0 2009-12-01 07:45:00   6.95      13085.0  United Kingdom
1 2009-12-01 07:45:00   6.75      13085.0  United Kingdom
2 2009-12-01 07:45:00   6.75      13085.0  United Kingdom
3 2009-12-01 07:45:00   2.10      13085.0  United Kingdom
4 2009-12-01 07:45:00   1.25      13085.0  United Kingdom
Number of instances (rows): 525461
Number of attributes (columns): 8
Column names: ['Invoice', 'StockCode', 'Description', 'Quantity', 'InvoiceDate', 'Price', 'Customer ID', 'Country']
```

查看每列的缺失值并且丢弃：

```
# 2. 查看数据信息，包括每列的缺失值数量
missing_values_1 = df_1.isnull().sum()
missing_values_2 = df_2.isnull().sum()
print("Missing values in sheet1:\n", missing_values_1)
print()
print("Missing values in sheet2:\n", missing_values_2)
```

```
Missing values in sheet1:
 Invoice             0
StockCode           0
Description      2928
Quantity            0
InvoiceDate         0
Price               0
Customer ID    107927
Country             0
dtype: int64
```

```python
# 3. 删除有缺失值的行
num_rows_before_1 = df_1.shape[0]
num_rows_before_2 = df_2.shape[0]
df_1.dropna(inplace=True)
df_2.dropna(inplace=True)

num_rows_after_1 = df_1.shape[0]
num_rows_after_2 = df_2.shape[0]
print("Year 2009-2010 dataframe:")
print(f"Number of rows before dropping missing values: {num_rows_before_1}")
print(f"Number of rows after dropping missing values: {num_rows_after_1}")

print()
print("Year 2010-2011 dataframe:")
print(f"Number of rows before dropping missing values: {num_rows_before_2}")
print(f"Number of rows after dropping missing values: {num_rows_after_2}")

print()
print("检查一下:")
missing_values_1 = df_1.isnull().sum()
missing_values_2 = df_2.isnull().sum()
print("Missing values in sheet1:\n", missing_values_1)
print()
print("Missing values in sheet2:\n", missing_values_2)
print("目前无缺失值")
```

打印删除前后数据差异并且检查是否完全删除缺失值:

```
Year 2009-2010 dataframe:
Number of rows before dropping missing values: 525461
Number of rows after dropping missing values: 417534

Year 2010-2011 dataframe:
Number of rows before dropping missing values: 541910
Number of rows after dropping missing values: 406830

检查一下:
Missing values in sheet1:
 Invoice        0
StockCode      0
Description    0
Quantity       0
InvoiceDate    0
Price          0
Customer ID    0
Country        0
dtype: int64
```

删除已取消的交易（发票编号代码以 C 开头）:

```python
# 4. 确定并删除已取消的交易

print("Year 2009-2010 dataframe:\n")
num_rows_before_1 = df_1.shape[0]
print(f"Number of rows before dropping cancelled transactions: {num_rows_before_1}")
cancelled_transactions_1 = df_1[df_1['Invoice'].astype(str).str.contains('C', na=False)]
print("Year 2009-2010 Cancelled transactions count:", cancelled_transactions_1.shape[0])
df_1 = df_1[~df_1['Invoice'].astype(str).str.contains('C', na=False)]
num_rows_after_1 = df_1.shape[0]
print(f"Number of rows after dropping cancelled transactions: {num_rows_after_1}")
print()
```

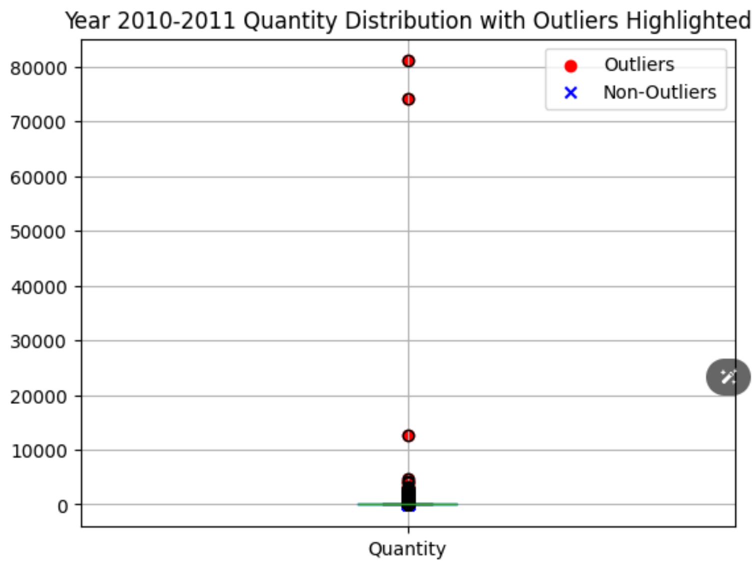
```
Year 2009-2010 dataframe:

Number of rows before dropping cancelled transactions: 417534
Year 2009-2010 Cancelled transactions count: 9839
Number of rows after dropping cancelled transactions: 407695
```

异常值处理:

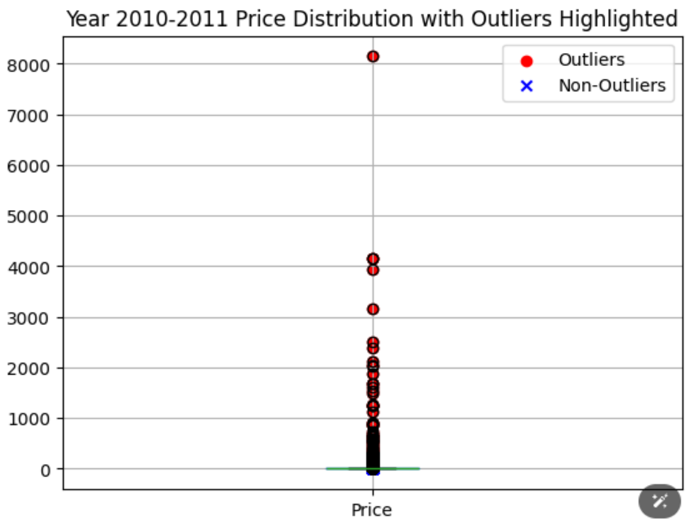
计算分位数: 对每个指定的列（"Quantity"和"Price"），计算 1%和 99%的分位数并打印;
标记非异常值和异常值: 使用 1%和 99%的分位数作为标准，筛选出每列中位于这两个分位数之间的数据，认为它们是正常的数据点。异常值: 选出每列中小于 1%分位数或大于 99%分位数的数据，认为这些是异常值;
打印异常值数量: 统计并打印出每列中被识别为异常值的数据点数量;
绘制箱线图并叠加异常值点: 箱线图能够直观地显示数据的中位数、四分位数及异常值;
绘制箱线图: 用红色圆点标记异常值，用蓝色叉号标记非异常值
裁剪异常值: 对于"Quantity"和"Price"两列中的每个数据点，如果它超出了 1%和 99%分位数的范围，则将其值设置为相应的边界值。这意味着小于 1%分位数的值会被设置为 1%分位数的值，大于 99%分位数的值会被设置为 99%分位数的值;
检查裁剪后的异常值数量: 理论上，由于数据已被裁剪到 1%和 99%的分位数，裁剪操作后的异常值数量应为 0

```
Year 2010-2011 percentiles in Quantity: [  1. 120.]
Number of outliers in Quantity: 3896
```




```
Year 2009-2010 Number of outliers in Quantity after clipping: 0
```

```
Year 2010-2011 percentiles in Price: [ 0.21 14.95]
Number of outliers in Price: 6766
```




```
Year 2009-2010 Number of outliers in Price after clipping: 0
```

删除 StockCode 和 Description 没有一一对应的数据行:

```
# 6. 确保StockCode和Description的一一对应性，并删除不符合条件的数据
print("Year 2009-2010 dataframe:\n")
unique_desc_1 = df_1.groupby('StockCode').Description.nunique()
non_unique_stock_codes_1 = unique_desc_1[unique_desc_1 > 1].index.tolist()
non_unique_descriptions_1 = df_1[df_1['StockCode'].isin(non_unique_stock_codes_1)]
num_rows_before_1 = df_1.shape[0]
print("Number of rows before removing non-unique StockCode entries:", num_rows_before_1)
print("Year 2009-2010 Number of rows with non-unique StockCodes:", non_unique_descriptions_1.shape[0])
print("Year 2009-2010 Non-unique StockCodes and their Descriptions:\n", non_unique_descriptions_1)
df_1 = df_1[~df_1['StockCode'].isin(non_unique_stock_codes_1)]
# 打印删除后的数据行数
num_rows_after_1 = df_1.shape[0]
print("Number of rows after removing non-unique StockCode entries:\n", num_rows_after_1)
```

```
Year 2009-2010 dataframe:

Number of rows before removing non-unique StockCode entries: 407695
Year 2009-2010 Number of rows with non-unique StockCodes: 87045
Year 2009-2010 Non-unique StockCodes and their Descriptions:
       Invoice StockCode                        Description  Quantity  \
2       489434    79323W                 WHITE CHERRY LIGHTS        12
7       489434     21523  FANCY FONT HOME SWEET HOME DOORMAT        10
8       489435     22350                            CAT BOWL        12
9       489435     22349      DOG BOWL , CHASING BALL DESIGN        12
10      489435     22195        HEART MEASURING SPOONS LARGE        24
...        ...       ...                                 ...       ...
525434  538171     21156             RETROSPOT CHILDRENS APRON         1
525435  538171    47591D        PINK FAIRY CAKE CHILDRENS APRON         1
525436  538171    47591B            SCOTTIES CHILDRENS APRON         2
525437  538171     22899          CHILDREN'S APRON DOLLY GIRL          1
525441  538171     22837           HOT WATER BOTTLE BABUSHKA          2

               InvoiceDate  Price  Customer ID         Country
2      2009-12-01 07:45:00   6.75      13085.0  United Kingdom
7      2009-12-01 07:45:00   5.95      13085.0  United Kingdom
8      2009-12-01 07:46:00   2.55      13085.0  United Kingdom
9      2009-12-01 07:46:00   3.75      13085.0  United Kingdom
10     2009-12-01 07:46:00   1.65      13085.0  United Kingdom
...                    ...    ...          ...             ...
525434 2010-12-09 20:01:00   1.95      17530.0  United Kingdom
525435 2010-12-09 20:01:00   1.95      17530.0  United Kingdom
525436 2010-12-09 20:01:00   1.65      17530.0  United Kingdom
525437 2010-12-09 20:01:00   2.10      17530.0  United Kingdom
525441 2010-12-09 20:01:00   4.65      17530.0  United Kingdom

[87045 rows x 8 columns]
Number of rows after removing non-unique StockCode entries:
 320650
```

stock code 中的 POST 表示邮费，并不代表产品，将其删除

```
# 7. 删除StockCode为'POST'的行
print("Year 2009-2010 dataframe:\n")
num_rows_before_removal_1 = df_1.shape[0]
print(f"Number of rows before removing 'POST' entries: {num_rows_before_removal_1}")
num_post_entries_1 = df_1[df_1['StockCode'] == 'POST'].shape[0]
print(f"Number of rows with 'StockCode' as 'POST' in Year 2009-2010: {num_post_entries_1}")
df_1 = df_1[df_1['StockCode'] != 'POST']
num_rows_after_removal_1 = df_1.shape[0]
print(f"Number of rows after removing 'POST' entries: {num_rows_after_removal_1}")
print()
```

```
Year 2009-2010 dataframe:

Number of rows before removing 'POST' entries: 320650
Number of rows with 'StockCode' as 'POST' in Year 2009-2010: 738
Number of rows after removing 'POST' entries: 319912
```

第二部分：Preparing Invoice-Product Matrix for ARL Data Structure

将原始数据转换为适合进行关联规则分析的格式，其中每一行代表一笔交易，每一列代表一个产品，由于 StockCode 和 Description 是一一对应的关系且都表示唯一的产品，所以我选择了看起来更直观的 Description。单元格的值表示该产品在该笔交易中是否存在（1 表示存在，0 表示不存在）。

由于数据量巨大，所以只需要选择一个国家，本次实验中我选择的是 **Korea**，该字段只在 Year 2009-2010 出现，所以只需要加载 df_1。由于内存限制，我无法把 48 个产品全都纳入计算，因此我将产品数量限制在"销量最高的前 13 个"中。

```python
import pandas as pd
from mlxtend.frequent_patterns import apriori, association_rules

# 假设df_1已经加载并进行了必要的预处理，筛选'Korea'的数据
df_1_Korea = df_1[df_1['Country'] == 'Korea']
print(f"Number of entries in df_1 for Korea: {df_1_Korea.shape[0]}")

# 计算每个产品的总销量并选择销量排名前13的产品
top_products = df_1_Korea.groupby('Description')['Quantity'].sum().sort_values(ascending=False).head(13).index

# 根据销量排名前13的产品创建一个发票-产品矩阵
basket = (df_1_Korea[df_1_Korea['Description'].isin(top_products)]
          .groupby(['Invoice', 'Description'])['Quantity']
          .sum().unstack().reset_index().fillna(0)
          .set_index('Invoice'))

# 将数量转换为1或0的函数
def encode_units(x):
    if x <= 0:
        return 0
    if x >= 1:
        return 1

# 应用encode_units函数到basket
basket_sets = basket.applymap(encode_units)

# 打印basket_sets的实例和属性数量
print(f"Number of instances (rows) in basket_sets: {basket_sets.shape[0]}")
print(f"Number of attributes (columns) in basket_sets: {basket_sets.shape[1]}")

# 打印basket_sets的前几行以验证结果
basket_sets.head()
```

得出的数据大小如下:

```
Number of entries in df_1 for Korea: 49
Number of instances (rows) in basket_sets: 2
Number of attributes (columns) in basket_sets: 13
```

Description Invoice	CACTI T-LIGHT CANDLES	CITRONELLA CANDLE GARDEN POT	DISCO BALL CHRISTMAS DECORATION	FROG CANDLE	HEART T-LIGHT HOLDER	MIRRORED WALL ART POPPIES	MIRRORED WALL ART SKULLS	MIRRORED WALL ART SNOWFLAKES	MIRRORED WALL ART SPLODGES	PACK 3 BOXES CHRISTMAS PANNETONE	ROTATING SILVER ANGELS T-LIGHT HLDR	SET OF 20 VINTAGE CHRISTMAS NAPKINS	TROPICAL HONEYCOMB PAPER GARLAND
522570	0	0	0	1	1	0	0	0	0	0	0	0	0
535831	1	1	1	0	0	1	1	1	1	1	1	1	1

第三部分：Determination of Association Rules

●利用 apriori 函数从购物篮数据中计算出频繁项集，即找出那些经常一起出现在交易中的商品组合。通过设置 min_support=0.01，只有那些至少在 1%的交易中出现的项集才被认为是频繁的。

●基于上一步得到的频繁项集，使用 association_rules 函数生成关联规则。这里使用的指标是支持度（metric="support"），且只考虑支持度至少为 1%的规则。

```
from mlxtend.frequent_patterns import apriori, association_rules
# 使用Apriori算法计算支持度，min_support设置为0.01
frequent_itemsets = apriori(basket_sets, min_support=0.01, use_colnames=True)
# 从频繁项集中生成关联规则
rules = association_rules(frequent_itemsets, metric="support", min_threshold=0.01)
# 排序并查看支持度最高的前五个关联规则
top_rules = rules.sort_values(by="support", ascending=False).head(5)
print(top_rules)
```

●将生成的关联规则按支持度从高到低排序，并提取支持度最高的前五条规则。

```
                                   antecedents  \
173053  (TROPICAL  HONEYCOMB PAPER GARLAND )
0                      (CACTI T-LIGHT CANDLES)
1               (CITRONELLA CANDLE GARDEN POT)
2            (DISCO BALL CHRISTMAS DECORATION)
3                      (CACTI T-LIGHT CANDLES)

                                               consequents  antecedent support  \
173053  (MIRRORED WALL ART SPLODGES, SET OF 20 VINTAGE...                  0.5
0                            (CITRONELLA CANDLE GARDEN POT)                  0.5
1                                   (CACTI T-LIGHT CANDLES)                  0.5
2                                   (CACTI T-LIGHT CANDLES)                  0.5
3                         (DISCO BALL CHRISTMAS DECORATION)                  0.5

        consequent support  support  confidence  lift  leverage  conviction  \
173053                 0.5      0.5         1.0   2.0      0.25         inf
0                      0.5      0.5         1.0   2.0      0.25         inf
1                      0.5      0.5         1.0   2.0      0.25         inf
2                      0.5      0.5         1.0   2.0      0.25         inf
3                      0.5      0.5         1.0   2.0      0.25         inf

        zhangs_metric
173053            1.0
0                 1.0
1                 1.0
2                 1.0
3                 1.0
```