

金融数据挖掘课设作业3 — 关联分析

题目：银行客户金融产品组合挖掘

本作业用 Apriori 和 FP-Growth 两种算法，对银行客户的产品持有数据做频繁项集挖掘和关联规则分析，并结合业务场景做解释。

一、数据读取与预处理

读取数据，看一下前 5 行、样本量和字段数量。

```
In [1]: import pandas as pd
import matplotlib.pyplot as plt
import networkx as nx
import time
import warnings
warnings.filterwarnings('ignore')

from mlxtend.preprocessing import TransactionEncoder
from mlxtend.frequent_patterns import apriori, fpgrowth, association_rules

# 让图里的中文正常显示
plt.rcParams['font.sans-serif'] = ['SimHei', 'Microsoft YaHei']
plt.rcParams['axes.unicode_minus'] = False

df = pd.read_csv('bank_product_association_data.csv')
print('样本量: ', df.shape[0], '条')
print('字段数: ', df.shape[1], '个')
df.head()
```

样本量： 2600 条
字段数： 9 个

Out[1]:

	customer_id	customer_segment	age	age_group	city_tier	income_level	risk_level
0	BA00001	稳健储蓄客户	45	中年	二线城市	较高	低
1	BA00002	青年薪资客户	24	青年	二线城市	中	中
2	BA00003	稳健理财客户	60	老年	二线城市	较高	中
3	BA00004	青年薪资客户	31	中年	二线城市	中	低
4	BA00005	青年薪资客户	23	青年	三线城市	中	中

说明：

- `customer_id` 只是客户编号，**不作为关联分析的项目**，这里只用它区分每一行客户。
- `transaction_items` 是用分号隔开的产品篮，是我们做关联分析的主要字段。
- 其它客户属性（`age_group`、`income_level`、`risk_level` 等）本身已经是离散类别（年龄 `age` 是连续值，但数据里已经给了离散化后的 `age_group`），这里主要分析产品之间的关联，所以不把属性放进购物篮。

下面把 `transaction_items` 转成 one-hot 的 0/1 矩阵（行是客户，列是产品）。

```
In [2]: # 把分号分隔的产品拆成列表
transactions = df['transaction_items'].str.split(';').tolist()

# 用 TransactionEncoder 转成 0/1 矩阵
te = TransactionEncoder()
te_arr = te.fit_transform(transactions)
basket = pd.DataFrame(te_arr, columns=te.columns_)

print('one-hot 矩阵形状：', basket.shape, '（行=客户，列=产品）')
print('产品种类数：', basket.shape[1])
basket.head()
```

one-hot 矩阵形状： (2600, 22) （行=客户，列=产品）
产品种类数： 22

Out[2]:

	保险产品	信用卡	信用卡分期	养老金账户	基金定投	基金申购	大额转账	定期存款	工资代发	房贷	...	活期储蓄	消费贷	第三方支付绑定
0	False	False	False	False	False	False	False	True	False	False	...	True	False	False
1	False	True	True	False	False	False	False	False	False	False	...	True	False	True
2	False	False	False	False	False	False	False	False	False	False	...	True	False	False
3	False	True	False	False	True	False	False	False	False	False	...	True	False	False
4	False	True	True	False	False	False	False	False	True	False	...	True	False	False

5 rows × 22 columns

二、Apriori 频繁项集挖掘

作业建议 `min_support` 不低于 0.06、`max_len` 不低于 3。先按 0.06 试一下。

```
In [3]: freq_006 = apriori(basket, min_support=0.06, use_colnames=True, max_len=3)
freq_006['长度'] = freq_006['itemsets'].apply(len)

for k in [1, 2, 3]:
    print('频繁 %d 项集个数：' % k, int((freq_006['长度'] == k).sum()))
```

频繁 1 项集个数: 17
频繁 2 项集个数: 79
频繁 3 项集个数: 132

发现问题: 频繁 1 项集只有 17 个, 没达到作业要求的 20 个。

原因: 本数据集一共只有 22 种产品, 其中“养老金账户、贷款逾期提醒、车贷、跨境汇款、投诉记录”这 5 种产品持有的人很少, 支持度低于 0.06, 所以在 0.06 下只剩 17 个频繁 1 项集。

按作业要求“数量不足时调低支持度”, 把 `min_support` 调低到 **0.03**, 这样能把“车贷(0.032)、贷款逾期提醒(0.047)、养老金账户(0.053)”也包含进来, 刚好达到 20 个。

```
In [4]: min_support = 0.03 # 从 0.06 调低到 0.03, 保证频繁 1 项集达到 20 个
max_len = 3

freq = apriori(basket, min_support=min_support, use_colnames=True, max_len=max_len)
freq['长度'] = freq['itemsets'].apply(len)

for k in [1, 2, 3]:
    print('频繁 %d 项集个数: ' % k, int((freq['长度'] == k).sum()))
print('频繁项集总数: ', len(freq))
```

频繁 1 项集个数: 20
频繁 2 项集个数: 103
频繁 3 项集个数: 230
频繁项集总数: 353

列出支持度最高的 10 个频繁项集。

```
In [5]: freq['商品组合'] = freq['itemsets'].apply(lambda x: ','.join(sorted(x)))
top10 = freq.sort_values('support', ascending=False).head(10)
top10[['商品组合', 'support', '长度']].reset_index(drop=True)
```

```
Out[5]:
```

	商品组合	support	长度
0	活期储蓄	1.000000	1
1	手机银行、活期储蓄	0.816538	2
2	手机银行	0.816538	1
3	工资代发、活期储蓄	0.407308	2
4	工资代发	0.407308	1
5	信用卡	0.390385	1
6	信用卡、活期储蓄	0.390385	2
7	网上银行	0.373846	1
8	活期储蓄、网上银行	0.373846	2
9	银行理财	0.372692	1

3 个有金融业务含义的产品组合解释：

1. **手机银行 + 活期储蓄**：支持度最高，说明几乎所有有活期储蓄的客户都开通了手机银行，是银行最基础的“账户 + 渠道”组合。
2. **工资代发 + 活期储蓄**：代发工资的客户基本都用活期账户收工资，这是典型的“薪资客群”，可以作为推销信用卡、理财的入口。
3. **网上银行 + 银行理财 + 活期储蓄**：愿意用网银的客户更倾向于买理财，属于有一定理财意识的客群，可以重点做财富管理推荐。

三、FP-Growth 频繁项集挖掘

用和 Apriori **相同的参数** (min_support=0.03, max_len=3) 再跑一次 FP-Growth，并记录两种算法的运行时间。

```
In [6]: # Apriori 计时
t1 = time.time()
freq_ap = apriori(basket, min_support=min_support, use_colnames=True, max_len=max_len)
t_ap = time.time() - t1

# FP-Growth 计时
t2 = time.time()
freq_fp = fpgrowth(basket, min_support=min_support, use_colnames=True, max_len=max_len)
t_fp = time.time() - t2

print('Apriori 频繁项集数: ', len(freq_ap), ' 用时: %.4f 秒' % t_ap)
print('FP-Growth 频繁项集数: ', len(freq_fp), ' 用时: %.4f 秒' % t_fp)
print('两种算法项集数量是否一致: ', len(freq_ap) == len(freq_fp))
```

Apriori 频繁项集数: 353 用时: 0.0157 秒

FP-Growth 频繁项集数: 353 用时: 4.5532 秒

两种算法项集数量是否一致: True

结果分析：

- 两种算法挖出的频繁项集**数量完全一致**，说明输入矩阵和参数都没问题，结果是可靠的。
- 在本数据集上 Apriori 反而更快。这是因为数据规模不大（2600 条、22 种产品），而且“活期储蓄、手机银行”这类产品几乎人人都有，数据比较**稠密**，FP-Growth 建 FP 树和递归挖掘的额外开销反而显得大。
- **FP-Growth 的效率优势主要体现在大规模数据上**：它只需要扫描两遍数据建树，之后在树上挖掘，不用像 Apriori 那样反复生成候选项集、反复扫描数据库。当产品种类多、交易量小时，Apriori 的候选爆炸会很严重，这时 FP-Growth 会明显更快。

四、关联规则生成与筛选

基于频繁项集生成关联规则，筛选条件：support \geq 0.06, confidence \geq 0.45, lift \geq 1.20。

```
In [7]: # 用 Apriori 的频繁项集生成规则
rules = association_rules(freq_ap, metric='confidence', min_threshold=0.45)
rules = rules[(rules['support'] >= 0.06) & (rules['lift'] >= 1.20)]

# FP-Growth 也生成一遍，对比规则数
rules_fp = association_rules(freq_fp, metric='confidence', min_threshold=0.45)
rules_fp = rules_fp[(rules_fp['support'] >= 0.06) & (rules_fp['lift'] >= 1.20)]

print('Apriori 筛选后规则数: ', len(rules))
print('FP-Growth 筛选后规则数: ', len(rules_fp))
```

Apriori 筛选后规则数: 265

FP-Growth 筛选后规则数: 265

剔除结构性规则：像“信用卡分期 → 信用卡”这种规则，业务上分期本来就是信用卡的功能，必然一起出现，意义不大，这里把它标注并剔除。

```
In [8]: def is_structural(row):
        a = set(row['antecedents'])
        c = set(row['consequents'])
        # 信用卡分期本身就属于信用卡，这种规则是结构性的
        if '信用卡分期' in a and c == {'信用卡'}:
            return True
        return False

rules['结构性规则'] = rules.apply(is_structural, axis=1)
print('被剔除的结构性规则数: ', int(rules['结构性规则'].sum()))
rules = rules[~rules['结构性规则']] # 剔除掉

# 整理成方便看的表格
rules['前项'] = rules['antecedents'].apply(lambda x: ','.join(sorted(x)))
rules['后项'] = rules['consequents'].apply(lambda x: ','.join(sorted(x)))
rules['前项数'] = rules['antecedents'].apply(len)
rules['后项数'] = rules['consequents'].apply(len)
rules = rules.sort_values('lift', ascending=False)
```

被剔除的结构性规则数: 5

挑选并展示有效规则，要求包含：≥4 条 1→1、≥4 条 2→1、≥2 条 1→2。

```
In [9]: cols = ['前项', '后项', 'support', 'confidence', 'lift']

r_1_1 = rules[(rules['前项数'] == 1) & (rules['后项数'] == 1)].head(5)
r_2_1 = rules[(rules['前项数'] == 2) & (rules['后项数'] == 1)].head(5)
r_1_2 = rules[(rules['前项数'] == 1) & (rules['后项数'] == 2)].head(3)

result = pd.concat([r_1_1, r_2_1, r_1_2])
print('1→1 规则: ', len(r_1_1), '条')
print('2→1 规则: ', len(r_2_1), '条')
print('1→2 规则: ', len(r_1_2), '条')
print('合计展示: ', len(result), '条')
result[cols].reset_index(drop=True).round(3)
```

1→1 规则： 5 条
2→1 规则： 5 条
1→2 规则： 3 条
合计展示： 13 条

Out[9]:

	前项	后项	support	confidence	lift
0	贵金属投资	证券账户	0.077	1.000	4.851
1	大额转账	证券账户	0.074	0.881	4.275
2	贵金属投资	基金申购	0.069	0.896	3.384
3	大额转账	基金申购	0.075	0.886	3.348
4	证券账户	基金申购	0.181	0.877	3.314
5	保险产品、信用卡	房贷	0.074	0.787	6.053
6	保险产品、工资代发	房贷	0.071	0.703	5.411
7	活期储蓄、贵金属投资	证券账户	0.077	1.000	4.851
8	基金申购、贵金属投资	证券账户	0.069	1.000	4.851
9	手机银行、贵金属投资	证券账户	0.064	1.000	4.851
10	房贷	保险产品、信用卡	0.074	0.568	6.053
11	房贷	保险产品、工资代发	0.071	0.547	5.411
12	贵金属投资	基金申购、证券账户	0.069	0.896	4.954

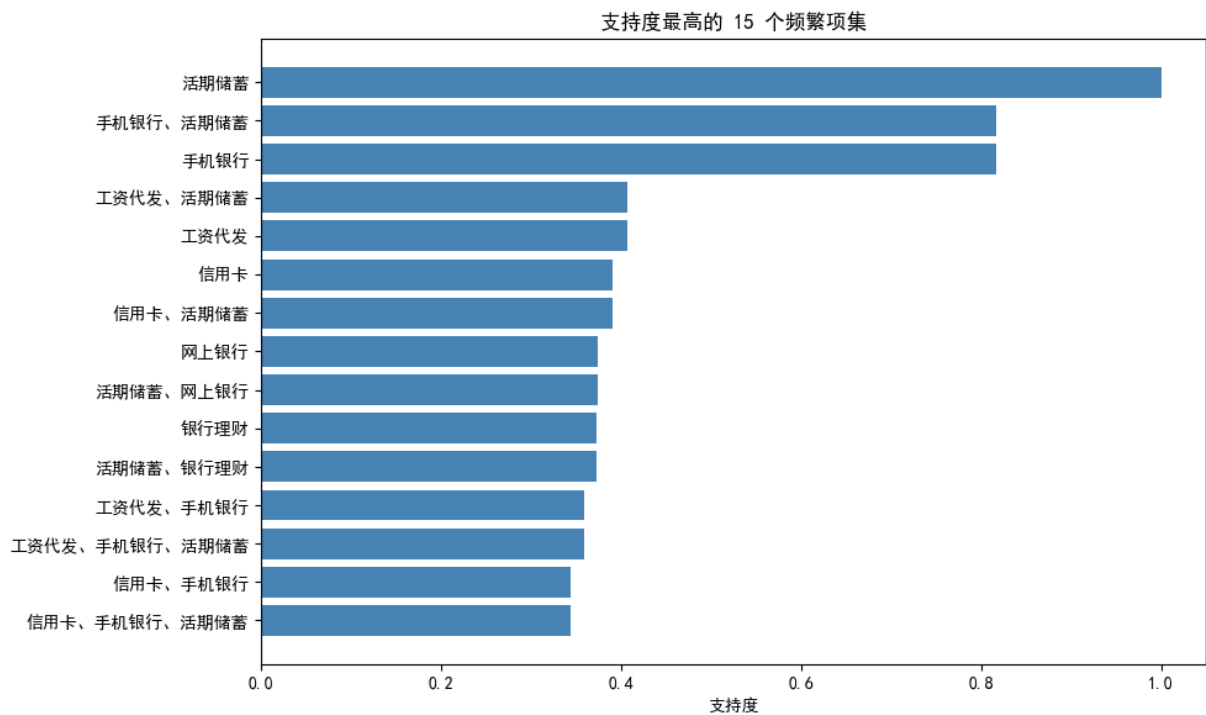
规则解读（举例）： lift 大于 1 说明前项和后项是正相关的，前项出现时后项更可能出现。比如“证券账户、贵金属投资 → 银行理财”这类规则，说明做证券和贵金属投资的客户往往也买理财，属于高净值客群，可以做交叉营销和财富管理推荐。

五、可视化

1. 频繁项集支持度柱状图

```
In [10]: top_plot = freq.sort_values('support', ascending=False).head(15)

plt.figure(figsize=(10, 6))
plt.barh(top_plot['商品组合'][::-1], top_plot['support'][::-1], color='steelblue')
plt.xlabel('支持度')
plt.title('支持度最高的 15 个频繁项集')
plt.tight_layout()
plt.show()
```



2. 关联规则网络图

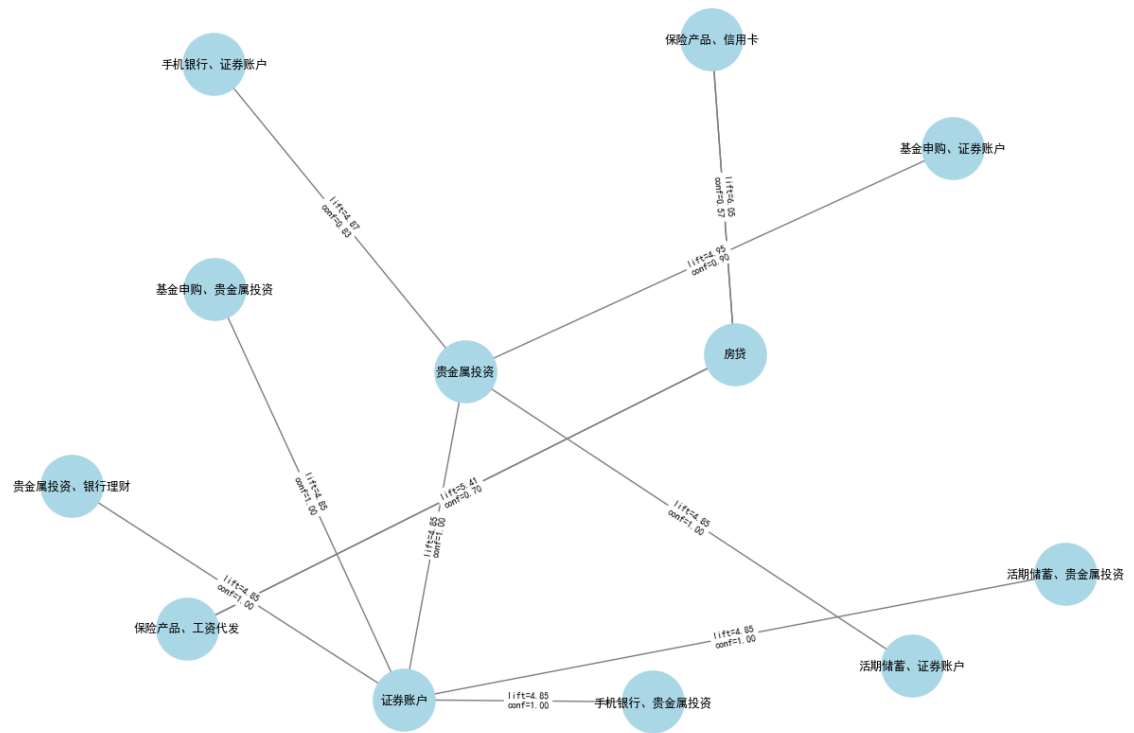
节点是产品或产品组合，箭头是 $A \rightarrow B$ 规则，边上标注 lift 和 confidence。这里取 lift 最高的 12 条规则来画。

```
In [11]: net_rules = rules.head(12)

G = nx.DiGraph()
for _, row in net_rules.iterrows():
    G.add_edge(row['前项'], row['后项'],
               label='lift=%.2f\nconf=%.2f' % (row['lift'], row['confidence']))

plt.figure(figsize=(13, 9))
pos = nx.spring_layout(G, k=1.5, seed=42)
nx.draw_networkx_nodes(G, pos, node_color='lightblue', node_size=2200)
nx.draw_networkx_labels(G, pos, font_size=9, font_family='SimHei')
nx.draw_networkx_edges(G, pos, arrowstyle='->', arrowsize=18, edge_color='gray')
edge_labels = nx.get_edge_attributes(G, 'label')
nx.draw_networkx_edge_labels(G, pos, edge_labels=edge_labels, font_size=7, font_family='SimHei')
plt.title('关联规则网络图 (边标签含 lift 和 confidence)')
plt.axis('off')
plt.tight_layout()
plt.show()
```

关联规则网络图（边标签含 lift 和 confidence）



六、小结

1. 数据预处理把产品篮转成了 one-hot 矩阵，customer_id 没有当成项目。
2. Apriori 在 0.06 下频繁 1 项集不够 20 个，调低到 0.03 后满足了所有数量要求。
3. Apriori 和 FP-Growth 挖出的频繁项集数量一致；本数据集较小且稠密，Apriori 更快，但 FP-Growth 在大规模数据上更有优势。
4. 关联规则筛选后剔除了结构性规则，得到的规则（如证券/理财相关组合）对交叉营销和财富管理推荐有参考价值。