基于网络模型的电商客户-商品关系分析

摘要

在本文当中构建了网络模型和个性化推荐模型，探究了客户与商品之间的关系，为后续商家更好的经营和销售提供了一定依据。

对于问题一, 首先进行数据预处理，对于异常值、无关值、重复值进行数据预处理，得到正确的 370998 条数据。对于客户购买行为的关联分析，本文利用 **FP-Growth 算法**构建频繁模式树，通过整体的频繁项集研究销售数量最多的前五个商品 **84077，22197，85099B，84879，85123A** 之间和其他商品之间的相关性进行说明分析。

对于问题二, 本文要求利用前五个月去预测第六个月的数据，以及整体的数据去预测新一个月的数据。因此本文建立 **ARIMA 模型**去进行预测，得到所有商品的预测数据。本文给出第六个月的前十个畅销商品的预测数据，新一个月前十个畅销商品的预测数据。第二小问要求的客户在一定金额之下购买第一问中最多的五件商品的最优模型，对此本文赋予商品一个得分 G_i, 其具体数值与我们预测所得的商品购买数量呈正相关。因此我们建立起以**完全背包问题**为基础的，以购买商品总得分最大化为优化目标的**线性优化模型**。

对于问题三, 本文利用 **RFM 综合评价模型**和**熵权法**综合给予每个客户一个得分，构建客户-商品矩阵，利用**余弦相似度**计算出来客户之间的相似性，给出相似性矩阵。本文基于相似性矩阵对不同的客户进行凝聚层次聚类，给出类别之中的网络模型和不同类别之间的网络模型，并且给出网络拓扑特征进行解释。

对于问题四, 根据问题三的关系网络模型，选择边最多的五个节点对应的客户作为最重要的客户，他们的客户 ID 为：**14911,15696,12949,13089,14769，**并选择这五个客户购买最多的前五个商品为重要商品：**22952,40016,84755,84949,22772**。最后针对客户个性化推荐本文利用商品-客户矩阵与商品相似性矩阵进行相乘得到**客户推荐商品矩阵**，从而建立个性化推荐模型，对客户进行个性化推荐。

关键字：FP-Growth 算法　ARIMA 模型　完全背包问题　线性优化模型　客户推荐商品矩阵

一、 问题重述

1.1 问题背景

随着互联网技术的快速发展，科学家、社会学家和经济学家需要分析和研究在线零售背后隐藏的一些规律和原理，通过不断优化网站设计、提高页面加载速度、简化购买流程等措施，用户体验得到显著提升，进而提升了转化率和用户满意度。越来越多的企业开始尝试全渠道布局，通过整合线上线下资源，提升购物体验和市场竞争力。

现有一数据集包含 2010 年 12 月 1 日至 2011 年 12 月 9 日期间发生的一家英国注册非商店在线零售商的所有交易，整个数据不包含缺失值，包含发票编号、变易代码、商品描述、数量、发票日期、单价、用户 ID 和国家等信息。有关这一数据集的研究主要是忽略商品描述信息后的关于用户或交易时间的分析，这将忽略商品购买中的有价值信息。而对一个在线零售商，分析哪些商品畅销、消费者的特征、商品的品牌影响力等是他更关心的问题。

1.2 问题要求

问题 1 通过客户的购买记录，请建立合适的数学模型找出客户购买行为的关联模式，并回答客户购买最多的前五个商品是什么，并对它们之间的相关性进行分析。并且，这五个商品对其他商品购买产生的影响是什么?

问题 2 利用前五个月的销售商品数据建模合适的模型预测第六个月的畅销商品有哪些? 再利用全部数据预测后一个月的畅销商品。给出某个客户（比如购买商品金额最多的客户）在一定的预算金额下购买第一问中找到的五件商品的最优模型。

问题 3 建立客户关系的网络模型。分析这个模型的相关拓扑特征，并给出合适的解释说明。

问题 4 基于第三问建立的客户关系网络模型，建模获得这个在线零售商最重要的前五个客户，及此网络中的五个商品，比较这五个商品与第一问的五个商品的差异性并给出相关解释。请建立数学模型帮助这个在线零售商针对热销商品的特定客户群体进行个性化推荐。

二、 问题分析

2.1 问题一分析

对于问题一，分析客户购买行为，首先进行数据预处理，清除异常和无关数据，然后使用 FP-Growth 算法构建频繁模式树，找到客户购买最多的前五个商品及其与其他商

品的关联，分析这些商品之间的相关性和购买模式。

2.2 问题二分析

对于问题二，使用 ARIMA 模型基于前五个月的销售数据预测第六个月的畅销商品，并扩展模型预测未来一个月的畅销商品。同时，建立一个以完全背包问题为基础的优化模型，最大化在一定预算内购买前五个最受欢迎商品的得分。

2.3 问题三分析

对于问题三，通过构建客户-商品矩阵，计算客户间的相似性，并利用这些相似性建立客户关系网络模型。分析网络的拓扑特征，如节点度、聚类系数和平均路径长度，来识别客户之间的潜在联系和分组。

2.4 问题四分析

对于问题四，基于客户关系网络模型，识别最重要的前五个客户及其购买的前五个商品，分析这些商品与问题一中的五个商品的差异性，并构建个性化推荐模型，通过商品-客户矩阵和商品相似性矩阵，为客户群体提供定制化的推荐。

三、 模型假设

为简化问题，本文做出以下假设：

- 假设 1，各个购买事件之间是独立的，即一个客户的购买行为不直接影响其他客户。
- 假设 2，客户关系网络在分析期间是静态的，客户的偏好和购物习惯在短期内不会发生显著变化。
- 假设 3，网络中识别出的五个重要商品与问题一中的五个商品有显著差异，这些差异反映了不同客户群体的偏好和需求。

四、 符号说明

符号	说明
G_i	购买第 i 个商品的得分
x_i	购买第 i 个商品的数量
w_i	购买第 i 个商品的单价
W	客户的预算金额
...	...

五、 问题一的模型的建立和求解

5.1 数据预处理

该数据集由 541，909 个条目和 8 列组成，记录了 2010 年 12 月 1 日至 2011 年 12 月 9 日期间发生的一家英国注册非商店在线零售商的所有交易，以下是每个专栏的简要概述：

· **InvoiceNo:** 发票编号，对象数据类型列，表示每个事务处理的发票编号。每个发票编号可以代表在一个交易中购买的多个项目。

· **StockCode:** 交易代码，对象数据类型列，表示每个项目的产品代码。

· **Description:** 描述，对象数据类型列，表示产品名称。它有一些缺失值，541，909 个条目中有 540，455 个非空条目。

· **Quantity:** 数量，整数列，包含每个事务处理中购买的产品数量。

· **InvoiceDate:** 发票日期，日期类型列，表示每个事务的日期和时间的 datetime 列。

· **UnitPrice:** 单价，浮点列，表示每个产品的单价。

· **CustomerID:** 客户 ID，浮点列，包含每个事务的客户 ID。该列有大量缺失值，541，909 个条目中只有 406，829 个非空条目。

· **Country:** 国家，对象数据类型列，记录每笔交易发生的国家的对象列。

分析可知，我们需要进行更深入的数据清理和预处理，以处理丢失的值和潜在的错误数据。

5.1.1 缺失值处理

CustomerID 列包含近四分之一的缺失数据。这个专栏对于聚集客户是必不可少的。插补如此大比例的缺失值可能会在分析中引入显著的偏倚或噪声。因此删除缺少 CustomerID 的行是保持聚类和分析完整性的最合理方法。

Description 列包含较小百分比的缺失值。然而，已经注意到数据中存在不一致的地方，相同的 StockCode 并不总是具有相同的 Description，这表明数据质量问题和产品描述中的潜在错误。鉴于这些不一致之处，根据 StockCode 估算缺失的描述可能不可靠。此外，由于缺失的百分比非常低，因此删除缺失 Description 的行以避免将错误和不一致传播到后续分析中同样也是最为合理的方法。

因此我们均采用删除缺失值的方法进行处理。

5.1.2 重复值处理

考虑在线零售数据中出现完全相同的行，包括相同的交易时间，表明这可能是数据记录错误，而不是真正的重复交易。保留这些重复行可能会在聚类中引入噪声和潜在的不准确性，因此我们删除这些完全相同的重复行。删除这些行将有助于实现更干净的数据集，这反过来又有助于根据客户独特的购买行为构建更准确的客户集群。

5.1.3 异常值处理

为了更好地了解客户的行为和偏好，我们需要考虑被取消的交易。根据题意，我们将通过过滤以“C”开头的行来识别这些事务。由于取消的交易中的所有数量都是负的，表明这些是取消的预期订单，因此我们只需要找出并将这些取消的订单合并入其原有订单中，就可以纠正订单的异常值。

5.2 FP-Growth 法分析关联模式

考虑客户购买行为的关联模式，我们采用 FP-Growth 进行分析。

FP-Growth 是一种关联分析算法，它采取分治策略，将提供频繁项集的数据库压缩到一棵频繁模式树，但仍保留项集关联信息。FP-Growth 算法基于以上的结构加快整个挖掘过程。相比 Apriori 等其他关联分析算法，FP-Growth 在性能上有一定优势，能够更快地挖掘出频繁项集和关联规则。其具体流程如下：

Step1：排序：根据频率对频繁项进行降序排序。

Step2：构建树：遍历事务数据库，对于每个事务，按照频繁项的顺序插入到 FP-tree 中。如果路径已经存在，则增加计数；如果路径不存在，则创建新的节点。

Step3：条件模式基：对于 FP-tree 中的每个频繁项，构建其条件模式基，即与该项相关的所有路径。

Step4：条件 FP-tree：基于条件模式基构建条件 FP-tree。

Step5：递归挖掘：在条件 FP-tree 上递归应用 FP-growth 算法，直到树为空或没有频繁项为止。

由此，我们将所有递归过程中发现的频繁项集合并，可以得到最终的频繁项集。

5.3 商品相关性及影响分析

首先，我们可以在经过数据预处理后的数据集中得到客户购买最多的五个商品，如下表所示：

表 1　购买最多的五个商品

商品描述	购买数量
84077	53215
22197	48712
85099B	45066
84879	35314
85123A	34204

根据得到的频繁项集，我们可以得出这五个商品的相关性如下所示：

（1）商品 84077 和商品 85123A 有很强的相关性且呈正相关关系，和其他三种商品呈现的关联度接近于 0，几乎不相关；

（2）商品 22197 和其他商品也几乎不相关；

（3）商品 84879 和其他四种商品相关性也接近 0；

（4）商品 85099B 与除了 85123A 以外的其他商品的相关性很低；

综上，只有商品 84077 与商品 85123A 有较强的正相关性，其他商品之间无关。

最后，考虑这五个商品对其他商品购买产生的影响，我们得出总体上前 5 个商品与部分商品有强相关性，即只对小部分商品购买产生的影响较强，对大部分商品购买影响较小。这也符合现实情况中商品购买行为的离散性。

六、 问题二的模型的建立和求解

6.1 数据预处理

问题二要预测的是畅销商品的情况，即商品的售卖数量，所以需要得到每种商品在每个月内的购买数量，只需要以 1 个月为单位，将里面的每种商品的销售量加起来，汇总成新的数据集即可。

我们需要预测的是后面一个月的畅销商品，根据前面几个月的数据进行预测，实际上只需要考虑前面几个月的畅销商品数量即可，一般正常情况下不会发生太大变动。所以为了减小预测的压力和提高预测模型的效率，只选取前几个月的畅销商品用来预测。特别地，为计算方便，我们将畅销商品定义为销量前 10% 的商品。

6.2 ARIMA 时间序列预测

问题二需要前五个月的销售商品数据预测第六个月的畅销商品，对此我们采用 ARIMA 时间序列预测方法，并对其采用自相关函数 ACF 检验。

ARIMA 模型是一种流行且广泛使用的时间序列预测统计方法。本题中我们采用差分自回归移动平均模型：ARIMA（p，d，q）来对第六个月的畅销商品进行预测，其中 AR 是自回归，p 是自回归项；MA 是移动平均，q 为移动平均项数，d 是时间序列成为平稳时所做的差分次数。下面对该模型进行说明：

AR：自回归模型，描述的是当前值与历史值之间的关系，用变量自身的历史时间数据对自身进行预测。p 阶自回归过程的公式为：

$$y_t = \mu + \sum_{i=1}^{p} \gamma_i y_{t-i} + \epsilon$$

其中 y_t 是当前值，μ 是常数项，P 是阶数,γ_i 是自相关系数，ϵ 是误差。

MA：移动平均模型，关注的是自回归模型中的误差项的累加。q 阶自回归过程的公式为：

$$y_t = \mu + \epsilon_t + \sum_{i=1}^{q} \theta_i \epsilon_{t-i}$$

由此得出 ARIMA 模型的公式为：

$$y_t = \mu + \sum_{i=1}^{p} \gamma_i y_{t-i} + \epsilon + \sum_{i=1}^{q} \theta_i \epsilon_{t-i}$$

其中 d 为时间序列成为平稳时所做的差分次数。特别地，在本题中我们将 p，q，d 均设为 1。

而在模型检验方面，ACF 是一个完整的自相关函数，可将有序的随机变量与其自身比较，反映了同一序列在不同时序的取值之间的相关性。公式为：

$$ACF_k = \rho_k = \frac{Cov(y_t, y_{t-k})}{Var(y_t)}$$

根据上述参数设定和检验方式，我们可以预测出第六个月的畅销商品，其可视化结果和对应的 PCA 检验图如下：

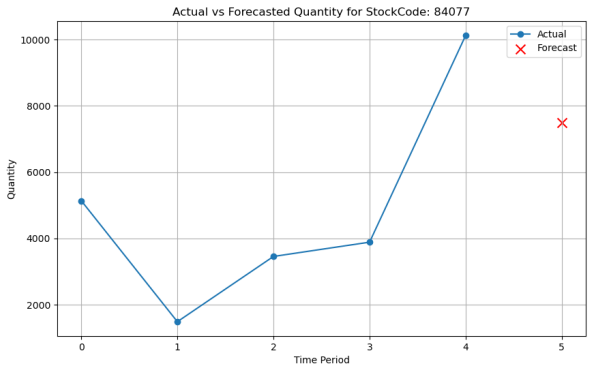


图 1　用前 5 个月的数据的预测结果

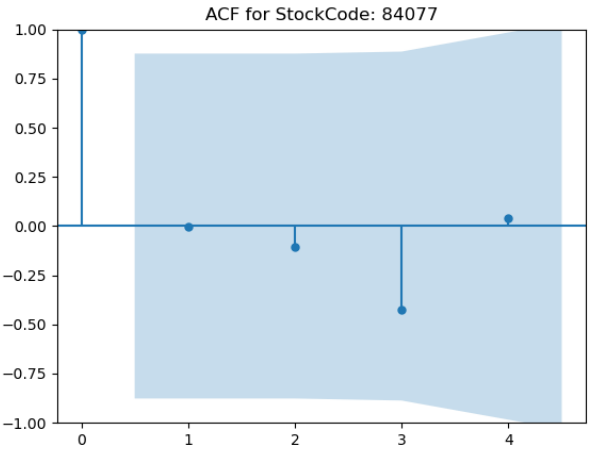


图 2　用前 5 个月数据预测的 ACF 值

得到的结果如下表:

表 2　第 6 个月的畅销商品情况

商品编号	销量	商品编号	销量
84077	7485	22084	3626
85099B	3468	17003	3318
22440	3068	21212	2990
22616	2779	85099F	2766
22386	2666	84755	2520

由此可见，该模型的预测效果良好，将其推广至利用全部数据预测后一个月的畅销商品，得到的可视化结果和对应的 PCA 检验图如下：

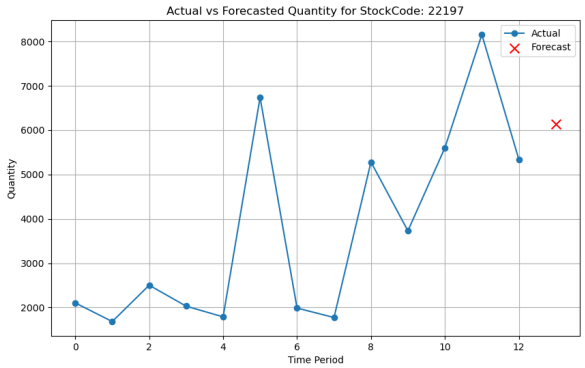


图 3　用全部数据的预测结果

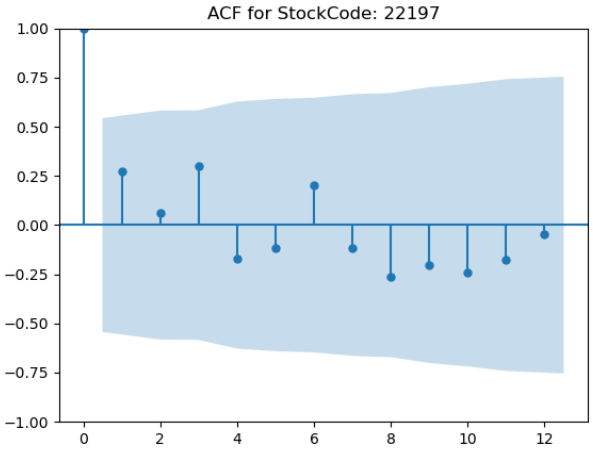


图 4　用全部数据预测的 ACF 值

得到的结果如下表：

表 3　新一个月的畅销商品情况

商品编号	销量	商品编号	销量
22197	6142	22086	4474
23084	4413	85099B	4406
22578	4252	84879	3774
84077	3670	85123A	3205
22910	2701	17003	2453

6.3 确定商品购买意愿

题目要求给出某个客户在一定的预算金额下购买第一问中所找到的物件商品的最优模型，我们需要在满足客户对商品购买意愿的前提下，使得商品的购买结果为最优。对此，我们赋予每个商品一个得分 G_i，其具体数值与我们预测所得的商品购买数量呈正相关。由此，对于第 i 个商品的购买量 x_i，我们可以得到：

目标函数：将五种热销商品按照求出的购买意愿大小，由大到小存放在集合 Y 中，i 表示的是商品在集合中的顺序。

目标是使得商品的购买总得分最大化：

$$max \sum_{i=1}^{n} x_i \cdot G_i$$

约束条件：

（1）该客户的购买商品的总金额不超过预算金额：

$$\sum_{i=1}^{n} x_i \cdot w_i \leq W$$

其中，W 表示该客户的预算金额，w_i 表示第 i 个商品的单价。

（2）每种商品的购买数量为大于等于 0 的整数：

$$x_i \geq 0, x_i \in Z$$

（3）要满足客户对商品的购买意愿：

$$x_1 \geq x_2 \geq ... \geq x_n$$

综上所述，该最优模型可以写成：

$$max \sum_{i=1}^{n} x_i \cdot G_i$$

$$\text{s.t.} \begin{cases} \sum_{i=1}^{n} x_i \cdot w_i \leq W \\ x_i \geq 0, x_i \in Z \\ x_1 \geq x_2 \geq ... \geq x_n \end{cases}$$

七、 问题三的模型的建立和求解

7.1 模型建立与求解

7.1.1 客户关系模型构建

在本题中，在问题三当中问题要求我们去考虑客户之间的联系，构建起客户之间的网络模型并且给出拓扑特征。因此我们首先利用熵权法结合 RFM 评价体系给不同的特征赋予一个权重，给予每个客户一个得分。RFM 是一种用于分析客户价值和细分客户群的方法。具体代表为：

新近购买度值（R）：此指标表示客户最近进行购买的时间。较低的新近度值意味着客户最近购买，表明与品牌的参与度较高。

频率（F）：此指标表示客户在一定时期内购买的频率。频率值越高，表明客户与企业的互动越频繁，这意味着忠诚度或满意度越高。

货币（M）：此指标表示客户在一定时期内花费的总金额。具有较高货币价值的客户对业务的贡献更大，这表明他们潜在的高终身价值。

有了上述指标，我们需要建立购买意愿评分模型，需要对消费者的新近购买度值、购买频率值、货币价值赋予权重来建立模型。在问题三当中问题除了要利用 RFM 评价体系赋予特征权重还需要考虑特征本身数值的特性，因此我们同时考虑利用熵权法进行特征权重赋值，下面是熵权法的步骤。

步骤 1：数据标准化处理

由于不同特征的量纲和取值范围不同，首先需要对数据进行标准化。假设有 m 个客户，每个客户有 n 个特征。特征矩阵表示为：

$$X = (x_{ij})_{m \times n}$$

其中，x_{ij} 表示第 i 个客户的第 j 个特征值。

标准化处理后的数据矩阵 $Y = (y_{ij})_{m\times n}$ 可通过如下公式进行处理：

$$y_{ij} = \frac{x_{ij} - \min(x_j)}{\max(x_j) - \min(x_j)}$$

其中，$\min(x_j)$ 和 $\max(x_j)$ 分别为特征 j 的最小值和最大值。

步骤 2：计算每个特征的比重

计算每个特征 j 在第 i 个客户的占比：

$$p_{ij} = \frac{y_{ij}}{\sum_{i=1}^{m} y_{ij}}$$

步骤 3：计算熵值

计算每个特征 j 的熵值 e_j：

$$e_j = -\frac{1}{\ln(m)} \sum_{i=1}^{m} p_{ij} \ln(p_{ij})$$

若 $p_{ij} = 0$，定义 $p_{ij} \ln(p_{ij}) = 0$（符合极限处理的定义）。

步骤 4：计算熵冗余度

计算每个特征的熵冗余度 d_j：

$$d_j = 1 - e_j$$

步骤 5：计算特征权重

根据每个特征的熵冗余度计算特征的权重 w_j：

$$w_j = \frac{d_j}{\sum_{j=1}^{n} d_j}$$

这些权重 w_j 就是我们需要的每个特征的重要性度量。得到权重过后我们对于每个客户建立以下的线性模型计算其得分，用于后续客户-商品得分矩阵元素的构建。由于客户的特征有购买商品的种类（品类数）、购买商品的数量（总数量）、购买商品的价格（总花费或单价）、购买商品的日期（可以转化为最近一次购买距今的天数）、客户的国家（可以编码成数值特征），因此我们分别用 T、N、P、D、C 来表示这五个客户特征，结合 RFM 评价方法和熵权法得到客户得分线性模型为：

$$S = w_1 * T + w_2 * N + w_3 * P + w_4 * D + w_5 * C$$

基于上述，我们可以通过客户得分构建起客户-商品得分矩阵，其列表示不同客户，行表示不同商品，元素则是客户商品得分。

CustomerID	StockCode1	StockCode2	StockCode3	$\cdots$	StockCodeN
Customer1	100	0	50	$\cdots$	30
Customer2	0	75	20	$\cdots$	0
Customer3	60	0	0	$\cdots$	15
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
CustomerM	10	40	0	$\cdots$	5

得到客户-商品得分矩阵过后我们利用余弦相似性去计算不同客户之间的相似度：

$$\text{Cosine Similarity}(\mathbf{A}, \mathbf{B}) = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|}$$

其中 A、B 为两个客户的购买行为向量，同时其中：

- $\mathbf{A} \cdot \mathbf{B}$ 是向量的点积（内积）：

$$\mathbf{A} \cdot \mathbf{B} = \sum_{i=1}^{n} a_i b_i$$

- $\|\mathbf{A}\|$ 是向量 **A** 的欧几里得范数（长度）：

$$\|\mathbf{A}\| = \sqrt{\sum_{i=1}^{n} a_i^2}$$

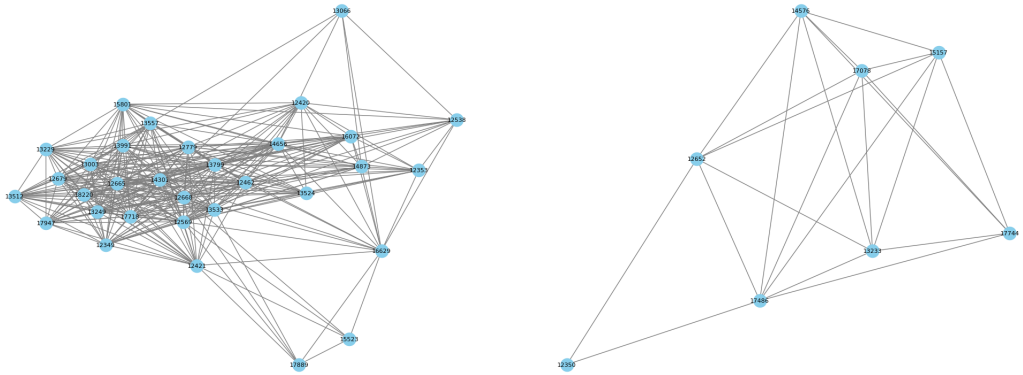
- $\|\mathbf{B}\|$ 是向量 **B** 的欧几里得范数（长度）：

$$\|\mathbf{B}\| = \sqrt{\sum_{i=1}^{n} b_i^2}$$

7.1.2 客户类别网络模型构建

得到相似度矩阵过后我们去进行凝聚层次聚类得到不同客户属于的不同的类别，针对不同差异性阈值的参数敏感性分析我们得到当阈值比较小，导致客户之间的余弦相似度只要很小就可以被分成一类，即客户的分类门槛很低，而且会导致分类的依据很模糊，所以不选取阈值很小的数；而随着阈值增大到一定数，客户之间的余弦相似度需要达到非常高，甚至有些客户无法和任一客户之间的相似度达到阈值，会自己形成以个聚类，我们将聚类里只有一个客户的，认为是与其他客户不存在任何关系的客户，不将这几类算入连通子图，所以数量在随阈值的增加而减小。所以我们选择 95% 作为差异值的区分，得到本文当中能够得到客户能够分成 156 类，因此我们对应的去构建不同网络之间的网络模型，其中

- V 是顶点集，代表所有的客户（即余弦相似度矩阵的行/列引索）。



(a) 第 16 类中的网络模型图　　**(b) 第 55 类中的网络模型图**

图 5　部分无向图网络模型

- E 是边集，如果两个客户 i 和 j 之间的相似度大于阈值 0.05，则存在一条边 (i, j)，边的权重是两者之间的相似度值。

绘制出的部分无向图如下：

通过不同类别当中的网络模型我们可以得到其拓扑特征如下

表 4　拓扑特征表

特征名	值	特征名	值
节点数	325	节点数	8
边数	325	边数	22
密度	0.6552	密度	0.7857
聚集系数	0.8278	聚集系数	0.8976

由于我们将整个网络进行聚类得到 156 类种不同兴趣的购买群体，因此我们再将这个 156 类不同群里之间的网络模型构建出来来判断不同类别之间的关系，下面是不同类别之间的网络模型图。

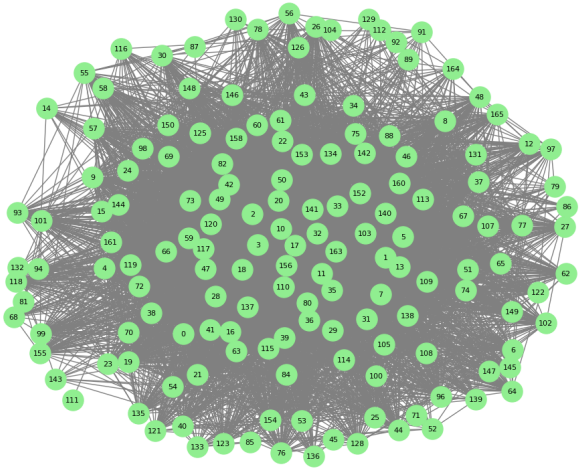


图 6　不同类别网络模型图

下面给出不同类别的网络拓扑特征：

表 5　拓扑特征表

特征名	值
节点数	156
边数	5720
密度	0.4731
聚集系数	0.7101

分析和解释拓扑特征：

（1）连通分量：一个连通分量是网络中一个最大的连通子图，其中的任意两个节点都可以通过边互相到达。该网络中存在多个连通分量，说明了客户之间的购买行为在某些方面是隔离的，反应了不同的购买群体，即我们一共分成了 156 种不同兴趣的购买群体。

（2）聚集系数：用来衡量网络中节点的聚集程度，即一个节点的邻居之间也互为邻居的比例。这里的聚集系数为 0.7101，为较高的聚集系数，表明客户之间的购买行为具有较强的社区性，即购买相似性商品的客户之间也倾向于有共同的购买行为。

八、 问题四的模型的建立和求解

8.1 识别最重要客户

我们将问题三中建立的客户关系的网络模型的节点中边最多的前五个节点所代表的客户视为是对于在线零售商最重要的前五个客户，具体理由说明如下：

（1）高连接性：边最多的客户在网络中与其他客户建立了广泛的联系，这种高连接性是的他们具有较高的影响力。也就是他们的购买行为或推荐往往会影响到更多的客户，从而带动商品的销售。

（2）忠诚度和复购率：边最多的客户与零售商之间一般建立了深厚的信任和依赖关系，他们更有可能成为忠实客户并持续复购。

（3）市场潜力：边最多的客户能够连接到的潜在客户数量也更多，这些潜在客户可能通过他们的推荐或社交互动了解到商品，并转化成为实际购买者。

综合上述几个原因，我们选择了以下几个客户作为在线零售商最重要的前五个客户以及最重要的五个商品如下表所示：

表 6　重要客户及商品情况

客户编号	商品编号
14911	22952
15696	40016
12949	84755
13089	84949
14769	22772

8.2 商品差异性分析

首先，我们给出两问中商品的差异性比较：

表 7　两问商品的差异性比较

题目	商品编号
问题一	80477,22197,85099B,84879,85123A
问题四	22952,40016,84755,84949,22772

由题意可知，这两个问题所选出的商品一个属于畅销商品，一个属于重要商品，我们对两者为什么存在差异进行如下的分析：

（1）关联性差异：分析两组商品在网络中的关联性差异。重要商品可能与其他商品有更紧密的联系，但畅销商品和其他商品之间的联系不一定紧密。

（2）客户差异：畅销商品通常受到广泛的普通客户群体的青睐，这些客户注重性价比，通常购买价格较低、需求普遍的商品。而重要商品可能吸引更具特定需求的客户群体，例如忠诚度高的老客户或对品牌有强烈偏好的客户，这些客户更倾向于购买价格较高、品牌影响力大的商品。

（3）购买行为：畅销商品的客户购买行为通常更加频繁，但单次购买量较小，且客户之间的联系较为分散。相比之下，重要商品的客户购买行为可能更具战略性，购买频次较低，但单次购买量较大，且这些客户之间的网络联系更加紧密。

（4）地理分布：重要商品的客户可能集中在特定的地区，形成区域性消费群体，这可能反映了这些商品在某些市场的独特吸引力。而畅销商品则可能在各个地区的销售都较为平均，表现出更广泛的市场覆盖。

（5）商品种类：畅销商品通常属于快消品类，如日常用品或低价格、需求频繁的商品。这些商品的生命周期较短，但销量大。相反，重要商品可能属于耐用品、奢侈品或高附加值的产品，这些商品在网络中可能占据了关键位置，虽然销量相对较少，但对客户的吸引力较强，且在客户关系网络中的影响较大。

九、 模型的评价

9.1 模型的优点

- 优点 1 问题二采用 ARIMA 时间序列进行预测，其参数具有明确的统计意义，可以用来解释时间序列数据的变化规律。这使得 ARIMA 模型在预测和决策方面具有较高的可信度和可解释性。
- 优点 2 问题三采用 RFM 模型对客户进行评价，可以快速识别出高价值客户、潜在客户和低价值客户等不同类型的客户，可以更好地了解客户需求和行为。

9.2 模型的缺点

- 缺点 1 问题二的预测未采取多种预测方式进行比较择优，可能得不到最优的预测效果。
- 缺点 2 问题四的优化模型建立存在较强的主观性。

参考文献

[1] 司守奎, 孙玺菁. 数学建模算法与应用[M]. 北京: 国防工业出版社, 2011.

[2] 卓金武. MATLAB 在数学建模中的应用[M]. 北京: 北京航空航天大学出版社, 2011.

附录 A　文件列表

文件名	功能描述
problem1.py	问题一程序代码
problem2.py	问题二程序代码
problem3.py	问题三程序代码
problem4.py	问题四程序代码

附录 B　代码

problem1.py

```python
import pandas as pd
from scipy.stats import pearsonr
from collections import Counter

# 读取Excel文件中所需的列
file_path = '1(1).xlsx'
data = pd.read_excel(file_path, usecols=["InvoiceNo","
   StockCode", "Quantity"])

# 过滤掉Quantity为负数或零的数据
data = data[data['Quantity'] > 0]

# 统计每个商品的总购买数量
item_counts = data.groupby('StockCode')['Quantity'].sum().
   sort_values(ascending=False)

item_counts
# 获取购买数量最多的前五个商品的商品代码
top_5_items = item_counts.head(5).index.tolist()

# 打印前五个商品及其购买数量
print("客户购买最多的前五个商品及其购买数量: ")
print(item_counts.head(5))
```

```
from mlxtend.frequent_patterns import apriori,
   association_rules
from  mlxtend.frequent_patterns import fpgrowth

# 假设 'data' 是一个包含 'InvoiceNo'（订单编号）和 'StockCode'
   （商品代码）的 DataFrame
# 将数据转换为适合 Apriori 算法的数据结构
basket = data.groupby(['InvoiceNo', 'StockCode'])['Quantity'].
   sum().unstack().reset_index().fillna(0).set_index('InvoiceNo
   ')
basket = basket.applymap(lambda x: 1 if x > 0 else 0)
# 使用FP-Growth算法寻找频繁项集
frequent_itemsets = fpgrowth(basket, min_support=0.01,
   use_colnames=True)

# 根据频繁项集生成关联规则
rules = association_rules(frequent_itemsets, metric='lift',
   min_threshold=1.0)

print(rules)
# # 获取支持度最高的前100条关联规则
# top_100_rules = rules.sort_values(by='support', ascending=
   False).head(100)

# # 显示结果
# print(top_100_rules)
# 提取规则中的前件和后件及其置信度
rules_matrix = rules.pivot(index='antecedents', columns='
   consequents', values='confidence')

# 填充缺失值为0
rules_matrix = rules_matrix.fillna(0)

# 显示矩阵
print(rules_matrix)
```

```
49
50 # 将矩阵写入Excel文件
51 output_file = '关联规则矩阵.xlsx'
52 rules_matrix.to_excel(output_file)
```

problem2.py

```
1  import pandas as pd
2  import numpy as np
3  import matplotlib.pyplot as plt
4  from statsmodels.tsa.arima.model import ARIMA
5  from statsmodels.graphics.tsaplots import plot_acf
6  import warnings
7  warnings.filterwarnings("ignore", message="Non-invertible
       starting MA parameters found")
8  warnings.filterwarnings("ignore", message="Non-stationary
       starting autoregressive parameters found")
9
10 file_path = '1.xlsx'
11 data = pd.read_excel(file_path, usecols=["StockCode", "
       Quantity", "InvoiceDate"])
12 data['YearMonth'] = pd.to_datetime(data['InvoiceDate']).dt.
       to_period('M')
13 monthly_sales = data.groupby(['StockCode', 'YearMonth'])['
       Quantity'].sum().reset_index()
14 monthly_sales = monthly_sales.set_index(['StockCode', '
       YearMonth']).unstack(fill_value=0).stack().reset_index()
15 monthly_sales = monthly_sales.sort_values(by=['StockCode', '
       YearMonth'])
16 quantity_sequences = monthly_sales.groupby('StockCode')['
       Quantity'].apply(list).reset_index()
17
18 print(quantity_sequences)
19
20 def predi(quantity_sequence, x):
21     best_stock_code = None
```

```
    best_forecast_stock_code = None
    max_forecast_value = -np.inf
    forecast_results = []

    for index, row in quantity_sequence.iterrows():
        stock_code = row['StockCode']
        quantity_series = pd.Series(row['Quantity'])

        if len(quantity_series) >= x:
            model = ARIMA(quantity_series[:x], order=(1, 1, 1))
            model_fit = model.fit()

            forecast = model_fit.forecast(steps=1)

            forecast_results.append({
                'StockCode': stock_code,
                'Forecast': int(max(forecast.tolist()[0], 0)),
            })

            if forecast.max() > max_forecast_value:
                max_forecast_value = forecast.max()
                best_stock_code = quantity_series[:x]
                best_forecast_stock_code = stock_code

    print(forecast_results)

    forecast_results_sorted = sorted(forecast_results, key=lambda x: x['Forecast'], reverse=True)
    top_10_forecast_results = forecast_results_sorted[:10]
    print("Top 10 forecasted values:")
    for result in top_10_forecast_results:
        print(f"StockCode: {result['StockCode']}, Forecast: {result['Forecast']}")

```

```
    if best_stock_code is not None:
        print(f"\nPlotting ACF for StockCode: {best_forecast_stock_code}")
        plt.figure(figsize=(10, 6))
        lags = min(20, len(best_stock_code) - 1)
        plot_acf(best_stock_code, lags=lags)
        plt.title(f"ACF for StockCode: {best_forecast_stock_code}")
        plt.show()

        plt.figure(figsize=(10, 6))
        plt.plot(range(len(best_stock_code)), best_stock_code, marker='o', label='Actual')
        plt.scatter(len(best_stock_code), max_forecast_value, color='red', marker='x', s=100, label='Forecast')
        plt.title(f"Actual vs Forecasted Quantity for StockCode: {best_forecast_stock_code}")
        plt.xlabel('Time Period')
        plt.ylabel('Quantity')
        plt.legend()
        plt.grid(True)
        plt.show()

predi(quantity_sequences, 5)
predi(quantity_sequences, 13)
```

problem3.py

```
import pandas as pd
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.cluster import AgglomerativeClustering

# 读取数据
file_path = '1.xlsx'  # 请确保文件路径正确
data = pd.read_excel(file_path)

```

```
# 构建客户-商品矩阵（以销售额表示）
data['Sales'] = data['Quantity'] * data['UnitPrice']
customer_item_matrix = data.pivot_table(index='CustomerID',
    columns='StockCode', values='Sales', aggfunc='sum',
                                            fill_value=0)

# 去掉全0的行和列
customer_item_matrix = customer_item_matrix.loc[
     (customer_item_matrix.sum(axis=1) > 0), (
    customer_item_matrix.sum(axis=0) > 0)]

# 计算客户之间的相似度（余弦相似度）
similarity_matrix = cosine_similarity(customer_item_matrix)

# 将相似度矩阵转为DataFrame形式
similarity_df = pd.DataFrame(similarity_matrix, index=
    customer_item_matrix.index, columns=customer_item_matrix.
    index)

print(similarity_df)

# 选择相似度大于等于0.05的作为合并条件，
    AgglomerativeClustering默认使用欧氏距离
# 1 - similarity_matrix 将相似度矩阵转换为距离矩阵（因为0.05的
    相似度相当于95%的差异）
clustering = AgglomerativeClustering(n_clusters=None, affinity
    ='precomputed', linkage='average',
                                        distance_threshold=0.95)
clusters = clustering.fit_predict(1 - similarity_matrix)

# 将聚类结果添加到客户数据中
customer_clusters = pd.DataFrame({'CustomerID':
    customer_item_matrix.index, 'Cluster': clusters})

# 过滤掉孤立点（即仅有一个客户的类）
```

```
cluster_counts = customer_clusters['Cluster'].value_counts()
non_isolated_clusters = cluster_counts[cluster_counts > 1].
    index
customer_clusters = customer_clusters[customer_clusters['
    Cluster'].isin(non_isolated_clusters)]

# 打印聚类结果
print(customer_clusters)

# 计算每个聚类的数量
cluster_counts = customer_clusters['Cluster'].value_counts()

# 获取聚类的总数（即不同的类别数量）
number_of_clusters = cluster_counts.count()

# 输出结果
print(f"聚类总数: {number_of_clusters}")

import pandas as pd
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.cluster import AgglomerativeClustering
import networkx as nx
import matplotlib.pyplot as plt
import seaborn as sns


# 构建同类客户的网络模型
def plot_intra_cluster_network(cluster_id):
    # 筛选同一类别的客户
    cluster_customers = customer_clusters[customer_clusters['
    Cluster'] == cluster_id]['CustomerID']

    # 构建子图
    G = nx.Graph()

```

```
    # 添加节点
    G.add_nodes_from(cluster_customers)

    # 添加边，基于相似度矩阵构建网络
    for i in cluster_customers:
        for j in cluster_customers:
            if i != j and similarity_df.loc[i, j] >= 0.05:  # 仅添加相似度高于0.05的边
                G.add_edge(i, j, weight=similarity_df.loc[i, j])

    # 绘制图形
    plt.figure(figsize=(10, 8))
    pos = nx.spring_layout(G)
    nx.draw(G, pos, with_labels=True, node_color='skyblue', edge_color='gray', node_size=300, font_size=8)
    plt.show()

    # 可视化同类客户的网络模型


for cluster_id in customer_clusters['Cluster'].unique():
    plot_intra_cluster_network(cluster_id)


# 聚合同类客户
def calculate_inter_cluster_similarity(customer_clusters, similarity_df):
    # 初始化类别相似度字典
    cluster_similarity = {}

    # 获取所有类别
    unique_clusters = customer_clusters['Cluster'].unique()

    # 遍历每个类别对
```

```
    for i in range(len(unique_clusters)):
        for j in range(i + 1, len(unique_clusters)):
            cluster_i = unique_clusters[i]
            cluster_j = unique_clusters[j]

            # 筛选出属于这两个类别的客户
            customers_i = customer_clusters[customer_clusters['Cluster'] == cluster_i]['CustomerID']
            customers_j = customer_clusters[customer_clusters['Cluster'] == cluster_j]['CustomerID']

            # 计算类别之间的平均相似度
            total_similarity = 0
            count = 0
            for customer_i in customers_i:
                for customer_j in customers_j:
                    total_similarity += similarity_df.loc[customer_i, customer_j]
                    count += 1

            if count > 0:
                avg_similarity = total_similarity / count
                cluster_similarity[(cluster_i, cluster_j)] = avg_similarity

    return cluster_similarity


# 计算类别之间的相似度
cluster_similarity = calculate_inter_cluster_similarity(customer_clusters, similarity_df)


# 构建跨类别的网络模型
def plot_inter_cluster_network(cluster_similarity):
```

```
    G = nx.Graph()

    # 添加类别节点
    clusters = list(set([c for pair in cluster_similarity.keys() for c in pair]))
    G.add_nodes_from(clusters)

    # 添加跨类别的边
    for (cluster_i, cluster_j), similarity in cluster_similarity.items():
        if similarity >= 0.01:  # 仅添加相似度高于0.05的边
            G.add_edge(cluster_i, cluster_j, weight=similarity)

    # 绘制图形
    plt.figure(figsize=(10, 8))
    pos = nx.spring_layout(G, k=0.6)
    nx.draw(G, pos, with_labels=True, node_color='lightgreen', edge_color='gray', node_size=500, font_size=8)

    plt.show()


# 绘制跨类别网络图
plot_inter_cluster_network(cluster_similarity)
```

problem4.py

```
import pandas as pd
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.cluster import AgglomerativeClustering
import networkx as nx
import matplotlib.pyplot as plt

# 读取数据
file_path = '1.xlsx'  # 请确保文件路径正确
```

```
data = pd.read_excel(file_path)

# 构建客户-商品矩阵（以销售额表示）
data['Sales'] = data['Quantity'] * data['UnitPrice']
customer_item_matrix = data.pivot_table(index='CustomerID',
    columns='StockCode', values='Sales', aggfunc='sum',
                                        fill_value=0)

# 去掉全0的行和列
customer_item_matrix = customer_item_matrix.loc[
     (customer_item_matrix.sum(axis=1) > 0), (
    customer_item_matrix.sum(axis=0) > 0)]

# 计算客户之间的相似度（余弦相似度）
similarity_matrix = cosine_similarity(customer_item_matrix)

# 将相似度矩阵转为DataFrame形式
similarity_df = pd.DataFrame(similarity_matrix, index=
    customer_item_matrix.index, columns=customer_item_matrix.
    index)

# 选择相似度大于等于0.05的作为合并条件，
    AgglomerativeClustering默认使用欧氏距离
clustering = AgglomerativeClustering(n_clusters=None, affinity
    ='precomputed', linkage='average',
                                     distance_threshold=0.95)
clusters = clustering.fit_predict(1 - similarity_matrix)

# 将聚类结果添加到客户数据中
customer_clusters = pd.DataFrame({'CustomerID':
    customer_item_matrix.index, 'Cluster': clusters})

# 过滤掉孤立点（即仅有一个客户的类）
cluster_counts = customer_clusters['Cluster'].value_counts()
non_isolated_clusters = cluster_counts[cluster_counts > 1].
```

```
    index
customer_clusters = customer_clusters[customer_clusters['
    Cluster'].isin(non_isolated_clusters)]


# 构建同类客户的网络模型并计算度数
def get_all_top_degree_nodes():
    all_degree_nodes = []

    # 遍历每个聚类
    for cluster_id in customer_clusters['Cluster'].unique():
        # 筛选同一类别的客户
        cluster_customers = customer_clusters[
    customer_clusters['Cluster'] == cluster_id]['CustomerID']

        # 构建子图
        G = nx.Graph()

        # 添加节点
        G.add_nodes_from(cluster_customers)

        # 添加边，基于相似度矩阵构建网络
        for i in cluster_customers:
            for j in cluster_customers:
                if i != j and similarity_df.loc[i, j] >= 0.05:
     # 仅添加相似度高于0.05的边
                    G.add_edge(i, j, weight=similarity_df.loc[
    i, j])

        # 计算每个节点的度数
        degree_dict = dict(G.degree())

        # 将该聚类的度数加入全局列表
        all_degree_nodes.extend(degree_dict.items())

```

```
    # 返回度数最高的五个节点
    top_5_nodes = sorted(all_degree_nodes, key=lambda x: x[1],
    reverse=True)[:5]

    return top_5_nodes


# 获取所有聚类中度数最高的五个节点
top_5_nodes = get_all_top_degree_nodes()

# 输出度数最高的五个节点
print(f"Top 5 nodes by degree across all clusters: {
   top_5_nodes}")

# 获取这五个节点（客户）的ID
top_5_customer_ids = [node[0] for node in top_5_nodes]

# 筛选出这些客户的购买记录
top_customers_data = data[data['CustomerID'].isin(
   top_5_customer_ids)]
top_5_item_ids = []
# 对每个客户分别计算购买最多的商品
for customer_id in top_5_customer_ids:
    customer_data = top_customers_data[top_customers_data['
   CustomerID'] == customer_id]
    item_counts = customer_data.groupby('StockCode')['Quantity
   '].sum()

    top_item = item_counts.idxmax()
    top_item_count = item_counts.max()
    top_5_item_ids.append(top_item)
    print(f"Customer {customer_id} purchased the most of
   StockCode {top_item} with {top_item_count} purchases")

top_items_data = data[data['StockCode'].isin(top_5_item_ids)]
```

```
96 print(top_items_data)
```