

2023 年第二届“钉钉杯”大学生 大数据挑战赛论文

题 目：智能手机用户监测数据分析

摘要

针对问题一，经过数据探索，发现原始数据存在一定的错误、冗余及缺失。数据清洗后，尝试从原始数据（数值、类别、时间）中提炼统计特征，以便于问题一和问题二的解决。

本节首先选取合适量化指标预处理，然后分别采用原型（K-Means++）、密度（DB-SCAN）、层次（AGNES）三种聚类算法对用户进行聚类，遵循“肘部法则”选择合理的聚类数量 K 值；最后，根据聚类结果对不同类别的用户画像，分析不同群体用户的特征。

关键词： TeX 图片 表格 公式

目录

一、 问题重述	1
1.1 问题背景	1
1.2 问题重述	1
二、 数据探索、预处理与特征提取	2
2.1 符号说明与数据概览	2
2.2 数据探索之类别变量	3
2.2.1 单一变量	3
2.2.2 天数变化	3
2.2.3 变量关联	4
2.3 数据探索之数值变量	5
2.3.1 单一变量	5
2.3.2 天数变化	6
2.3.3 变量关联	6
2.4 直觉小结与用户量化	7
2.4.1 数据直觉	7
2.4.2 用户模型	7
三、 问题一：聚类分析与用户画像	8
3.1 特征工程与评价指标	8
3.1.1 特征选择与数据降维	8
3.1.2 评价指标	9
3.2 算法概述与 K 值选择	9
3.2.1 原型聚类：K-Means++	10
3.2.2 层次聚类：BIRCH+AGNES	11
3.2.3 密度聚类：DBSCAN	12
3.3 算法比较与用户画像	13
3.3.1 算法比较	13
3.3.2 用户画像	14
四、 问题二：未来使用情况预测	18

4.1 评价指标与求解思路	18
4.1.1 评价指标	18
4.1.2 系统 Pipeline	19
4.2 超参数选择与预训练模型建立	21
4.2.1 建立 KNN 最近邻分类预测模型	21
4.2.2 建立 LogisticRegression 分类预测模型	21
4.2.3 建立随机森林集成分类预测模型	21
4.2.4 建立 XGBoost 集成回归预测模型	21
4.3 运行结果	22
4.3.1 运行结果	22
参考文献	23
附录 A 运行环境与环境依赖	25
附录 B 数据清洗、预处理源代码	25
附录 C 数据探索：数据概览源代码	29
附录 D 数据探索：类别变量代码	32
附录 E 数据探索：数值变量代码	36
附录 F 问题一：量化指标代码	42
附录 G 问题一：聚类算法代码	46
附录 H 问题一：用户画像代码	56
附录 I 问题二：训练资料代码	63
附录 J 问题二：分类问题代码	63
附录 K 问题二：回归问题代码	63
附录 L 问题二：模型评价代码	63

一、问题重述

1.1 问题背景

智能手机已成为现代社会人们生活不可或缺的一部分，其普及和发展给人们带来了巨大的生活便利和娱乐享受。近年中国智能手机市场品牌竞争进一步加剧，中国超越美国成为全球第一大智能手机市场。随着智能手机市场快速增长，智能手机用户群体愈发多样，智能手机软件满目琳琅，研究智能手机用户的行为模式和使用偏好对于理解用户需求、预测用户行为和优化产品与服务具有重要意义。通过对智能手机用户监测数据的分析，可以为智能手机制造商、软件开发者、广告商和营销人员等提供有益的信息及有价值的洞察，指导他们制定战略和决策，更好地贴合用户需求并提供更佳的用户体验。

1.2 问题重述

问题一 针对问题一，赛题要求 (1) 根据用户常用所属的 20 类 APP 的数据对用户聚类，
(2) 对不同类别的用户画像，分析不同群体用户的特征。

二、数据探索、预处理与特征提取

2.1 符号说明与数据概览

原始数据集包含 app 类别辅助表格(*A.app_class.csv*)与 21 天监测数据(*B*.dayxx.txt*), 来源、符号、意义及数据类型如下表 1 所示。

表 1 数据集原始特征

来源	符号	意义	类型
<i>A&B*</i>	<i>appid</i>	用户的 id, 唯一标识一名用户	类别变量
<i>A</i>	<i>app_class</i>	应用的 id, 唯一标识一个 APP	类别变量
<i>B*</i>	<i>app_type</i>	APP 类型: 系统自带、用户安装	类别变量
<i>B*</i>	<i>start_day</i>	使用起始天, 取值 1-30	数值变量
<i>B*</i>	<i>start_time</i>	使用起始时间	时间变量
<i>B*</i>	<i>end_day</i>	使用结束天	数值变量
<i>B*</i>	<i>end_time</i>	使用结束时间	时间变量
<i>B*</i>	<i>duration</i>	使用时长 (秒)	数值变量
<i>B*</i>	<i>up_flow</i>	上行流量	数值变量
<i>B*</i>	<i>down_flow</i>	下行流量	数值变量

将 *B** 与 *A* 进行“左连接”得到 *B* (同时舍弃重复值), *app_class* 为 a~t 的代表 *A* 中 20 个常用类别 (*B* 含 3931 种); *NaN* 则代表所属类别未知的不常用 APP (*B* 含 32506 种)。*B* 中, 在类别已知的常用 20 类 APP 中, *t* 类数量最多 (1406), *r* 类最少 (41)。

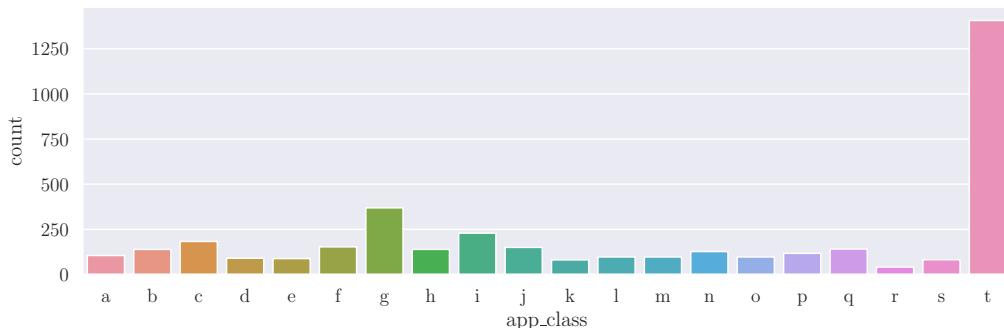


图 1 B 中各类 APP 计数图

2.2 数据探索之类别变量

2.2.1 单一变量

表 2 类别变量统计描述 (以 day01 为例)

	uid	appid	app_type	app_class
count	5335803	5335803	5335803	5335803
unique	35451	11021	4	21
top	A9E4AAC5B8E05D2A4E35E0D4F2994F37	3309	usr	NaN
freq	2629	924309	2987468	2432606

据表 2, *app_type* 只有两类【系统预装、用户安装】，存在异常。通过数据探索，发现表格存在 [’sys’, ’usr’, ’用户’, ’预装’] 四种取值，故将中文全部替换成英文。

app_class 有 21 类，这是因为在“左连接”操作时，将 *NaN* 也作为一种 APP 类型，这是由于此处数据缺失本身就代表一种资讯（小众 APP），并非随机发生或人为故意。如果将 *NaN* 也视作一种 *app_class*，那么数据 B 不存在缺失值。

2.2.2 天数变化

另外，从 21 天的类型变量数据可以发现，每日活跃用户、APP、日志条数在各天都有所差异，如图 2 所示。

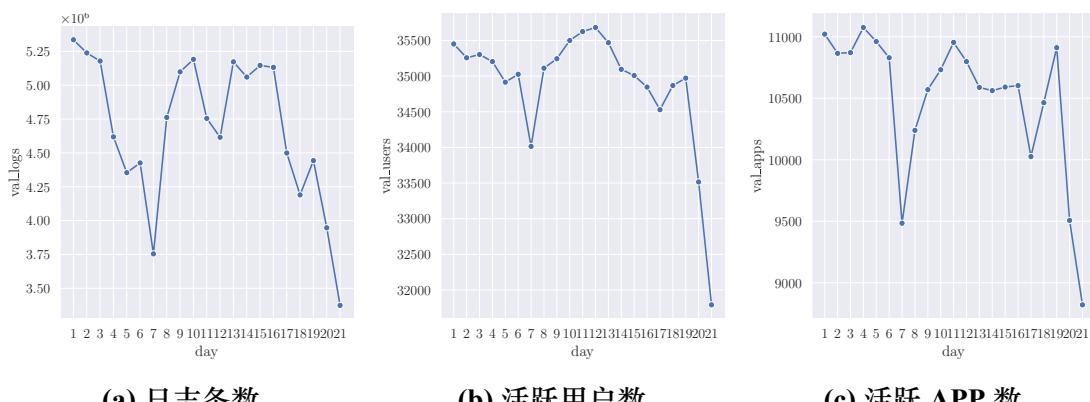


图 2 day01~day21 类别变量的变化折线图

粗略观察图 2b, 活跃用户数在 7 和 21 存在明显波谷，似乎和“星期”有某种关联；对照图 2a、图 2b、图 2c 三表分析，似乎 APP 活跃情况（种类、请求）与天数有所关联，甚至可以猜测某些小众 APP 被某些特定用户群体所使用甚至是青睐。

2.2.3 变量关联

APP 自身包含 *appid*、*app_class* 以及 *app_type* 属性，因此可以抽取这三列建立 *C*。

表 3 APP 统计描述

	<i>appid</i>	<i>app_type</i>	<i>app_class</i>
count	37276	37276	37276
unique	36437	2	21
top	19582	usr	NaN
freq	2	34153	32958

据表 3，数据探索发现有 839 个 app 多重类型，即对于某些用户而言，该软件为系统预装，对另一些而言，则为自行安装。这似乎表明，不同的用户在软件下载与安装层面，有可相互区分的行为特征。另外，21 类 APP 的数量情况如图 3 所示。

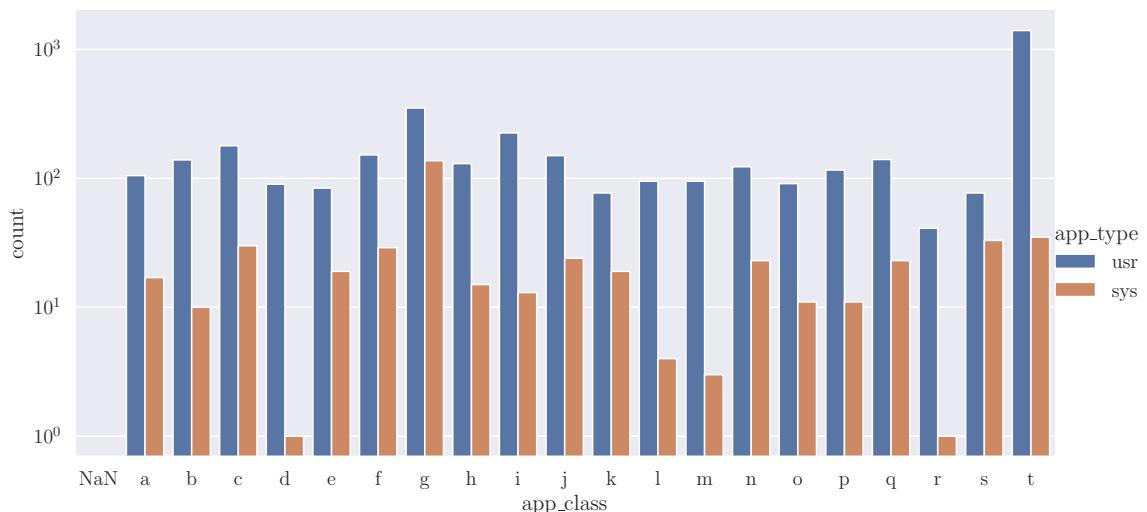


图 3 21 类 APP 计数图

据图 3，未分类的 APP 并不是小数目（有一部分为系统预装），*t* 类在 APP 多样性表现上依然出众。此外，不同种类的 APP 就类型（预装、用户）而言相差较为悬殊，例如：*g* 类，系统预装相对较多；其余类别，用户预装较为普遍，尤其是 *d* 和 *r* 类。

值得区分的是，在新建 APP 统计数据 *C* 中，计数图反映了各类 APP 的多样程度（市场垄断程度）。而在监测数据 *B* 中，某个（类）APP 请求日志（行数）计数图则反映了该用户在使用各类 APP 时的活跃（点击）行为，这将在下一小节进行探索。

2.3 数据探索之数值变量

2.3.1 单一变量

表 4 数值变量统计描述（以 day01 为例）

	<i>start_day</i>	<i>end_day</i>	<i>duration</i>	<i>up_flow</i>	<i>down_flow</i>
count	5335803	5335803	5335803	5335803	5335803
mean	0.975107	1	2151.604772	607572.168995	158163.759549
std	16.843899	0	1455335.155631	11015502.975274	6538529.614936
min	-16524	1	1	1	0
25%	1	1	3	0	0
50%	1	1	10	0	0
75%	1	1	36	1278	1063
max	1	1	1427769883	3639473769	3292713011

数据具体说明指出：“*start_day*: 使用起始天，取值 1-30（注：第一天数据的头两行的使用起始天取值为 0，说明是在这一天的前一天开始使用的）。”然而，表 4 显示其最小值为 -16542，因此可以判断 *start_day* 存在异常值；而这会直接导致 *duration*、*up_flow*、*down_flow* 偏差过大。因此，须要对这两列进行数据清洗，删除异常值。

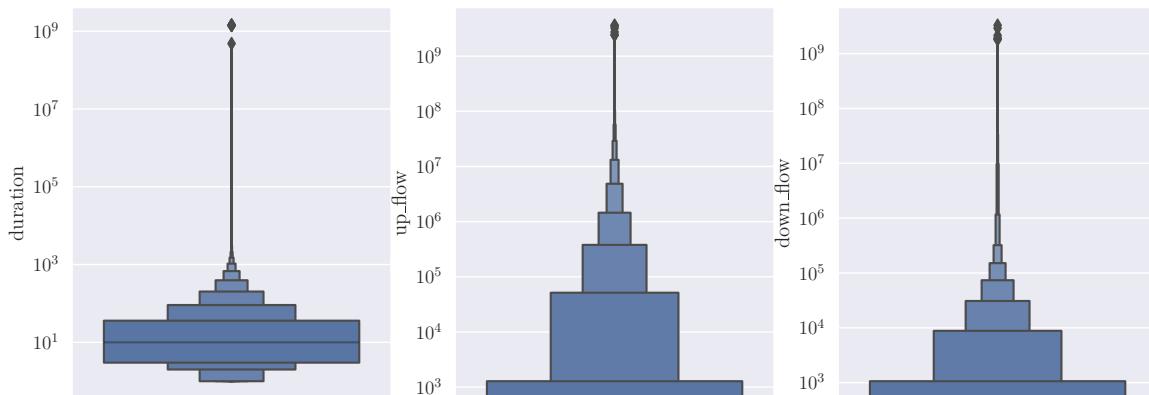


图 4 使用时长、上下行流量增强箱形图（以 day01 为例）

据统计，在第 1~21 天监测数据中，99.98% 的记录使用时长不超过 9158.02。针对异常案例进行分析，例如，*uid* = 64B3E40461C56847F35DB46D55707EA4 用户：

表 5 异常案例

appid	app_class	start_day	start_time	end_day	end_time	duration
4803	a	19	00:52:46	19	07:47:59	24912
18478	c	19	07:48:25	19	07:48:38	12
:	:	:	:	:	:	:
6192	NaN	19	20:46:11	19	20:46:29	18
3309	f	19	23:14:49	19	23:16:38	109

凌晨零时至早上七时的记录是不符合生活常态不可持续的，猜测是应用后台驻留、系统故障或用户因故未关闭应用。不过，是否存在异常行为可以作为一个新的特征。因此，本文对持续时间超过 9159 的认定为无效使用时长，不能真实反映用户的行为特征。

此外，以第一天为例，不同用户在 *appid* 个数、*duration* 总时长、*up_flow*、*down_flow* 总流量、日志行数层面有所差异，分布如图 5 所示。

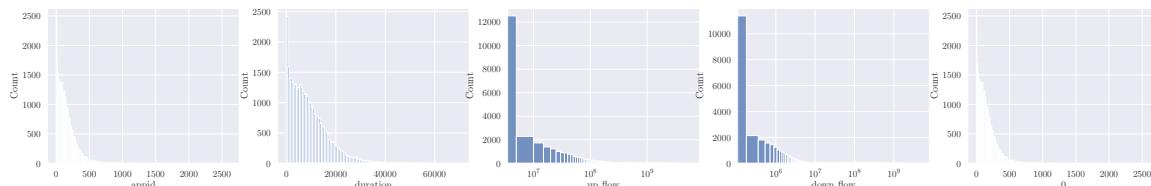


图 5 直方图（使用 APP 数量、使用有效总时长、消耗上下行流量、日志行数）

可以观察到某些用户较依赖手机，日志数、使用时长、APP 多样性、消耗流量偏多。

2.3.2 天数变化

探索发现，可为每名用户绘制使用 APP 总数/时长/流量/日志随天数变化的折线图，以展现用户在月（周）级别的变化趋势与独潜在规律，并据此将用户群归类。

2.3.3 变量关联

更细粒度地，对于每名用户/每天，可绘制其各类 APP 的个数/时长/流量/日志情况，以了解用户对不同类型 APP 的适用情况、青睐程度；更进一步，还可以绘制 24 小时各类 APP 使用情况，这样便于了解用户的作息、通勤、活跃时段等信息。

2.4 直觉小结与用户量化

2.4.1 数据直觉

概括来说，赛题数据 A 给出 4000 多个常用 APP 所属类别：这是 1 : 1 的数据；而赛题数据 B 则记录了每名用户 (uid) 每日每时使用各款 APP ($appid$) 的起始时间，使用时长，上下流量等信息：这是 1 : N 的资料。可以使用统计量进行归纳。

具体而言，监测数据蕴藏大量用户行为特征，例如：

- 用户一天使用多长时间的手机（可间接反映依赖程度、年龄）
- 用户平均多长时间看一次手机（可间接反映依赖程度、年龄、工作）
- 距离上一次上线，隔了几天（可间接反映依赖程度、年龄、工作）
- 在什么星期几的时间段最常用什么类型 APP（可反映工作、生活）
- 用户早、晚各使用什么类 APP（可反映工作、生活、作息）
- 用户周末、工作日各使用什么类 APP（可反映工作、生活、作息）
- 用户最早什么时候开始使用手机、什么时候结束使用（可反映工作、作息）
- 用户最常用什么类型 APP（可反映喜好）
- 哪一类 APP 使用最频繁、哪一类使用时长最多（可反映喜好）
- 用户一共安装了多少个 APP（可反映保守程度、对多样性的接纳程度）
- 系统预装与自行安装的比例（可反映保守程度、对多样性的接纳程度）
- 用户的每月流量使用情况（可间接反映财富程度、年龄）

2.4.2 用户模型

用户量化是指将现实生活中的“用户实体”进行抽象，采用不同维度的量化指标建模，即将其视为 n 维空间的一个点，使用形如 $X = [x_1, x_2, \dots, x_n]$ 的数学符号表示。

基于对数据集的深入探索及理解，提出简易用户模型：

表 6 简易用户模型

符号	意义	维度
uid	用户的唯一标识	1
$DU_{d,h,c}$	该用户在第 d 天 h 时内使用 c 类 APP 时，投入的总计时长	$d \times h \times c$
$UF_{d,h,c}$	该用户在第 d 天 h 时内使用 c 类 APP 时，消耗的上行流量	$d \times h \times c$
$DF_{d,h,c}$	该用户在第 d 天 h 时内使用 c 类 APP 时，消耗的下行流量	$d \times h \times c$
$NO_{d,h,c}$	该用户在第 d 天 h 时内使用 c 类 APP 时，记录的日志行数	$d \times h \times c$

三、问题一：聚类分析与用户画像

3.1 特征工程与评价指标

3.1.1 特征选择与数据降维

聚类指将数据样本对象划分成若干类（簇、标签）并尽可能的保证“类内紧凑”、“类间独立”[3]。不同的量化指标、不同的相似度量（距离定义），往往会带来迥异的聚类结果。一般来说，量化指标维度数目越多，算法运行时间越长、结论可解释性越弱。

关于用户画像的量化指标，陈 [5]、成 [6] 等人从日均屏幕使用时间切入；武 [7] 等人从APP 数量、阅读时间等特征对阅读类 APP 使用人群进行聚类解读；侯 [8] 针对每日手机使用时长、使用频次、使用偏好等特征对用户进行建模；韦 [9] 基于安装数量、打开次数、使用时长、工作日使用时长、周末使用时长构建用户特征。

首先，为避免数据泄漏，仅选择前 7 日各类 APP 总时长、频次、上下行流量作为量化特征，共计 $20 \times 4 = 80$ 维；对于不同的量纲特征，分别扣除均值，除以标准差以进行数值标准化。Pearson 相关热力图 6 显示，各类 APP 时长、频次、流量成弱正相关。

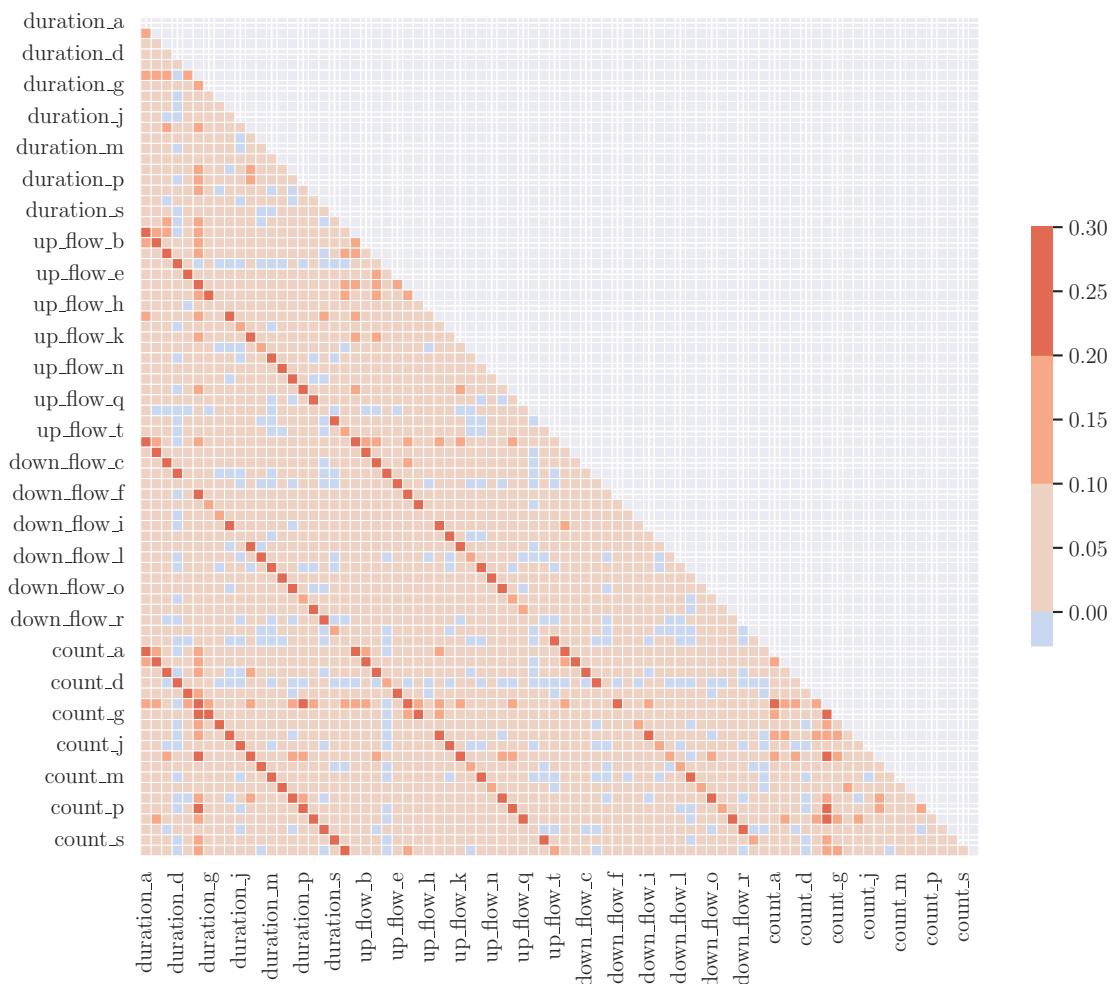


图 6 皮尔逊相关系数热力图

为增加数据易用性，降低计算开销，增强视觉理解，而后采用主成分分析对特征进行变换，并按方差排序表示各维度重要程度，如图 7，选定阈值将维度压缩至 3 维。

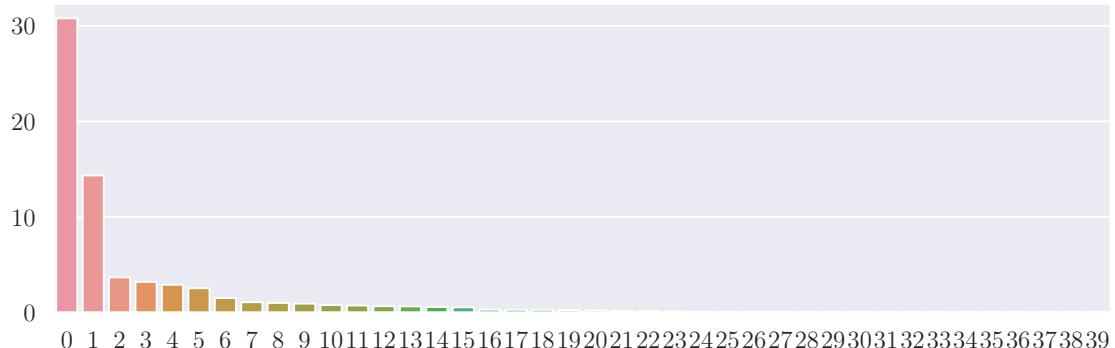


图 7 PCA 特征方差柱状图

3.1.2 评价指标

聚类“好坏”不存在绝对的客观的标准 [1]；聚类数目设定是否“合理”也往往依赖人工先验知识 [2]。聚类数目设定过低，划分粒度不够细腻；聚类数目设定过高，宏观结论的可解释性又受到限制。常用选择聚类数目方法是人为观察聚合系数折线图，大致估计最优聚类数量 K 。相关定义如下：

定义 1 各簇畸变程度：该簇重心与其内部成员位置距离的平方和；假设一共将 n 个样本划分到 K 个簇中，用 C_k 表示第 k 簇，该簇重心记为 u_k ，则第 k 簇的畸变程度为：

$$\sum_{i \in C_k} |x_i - u_k|^2 \quad (1)$$

定义 2 聚合系数：

$$J = \sum_{k=1}^K \sum_{i \in C_k} |x_i - u_k|^2 \quad (2)$$

此外，还有 Calinski-Harabasz 系数 [11]、Davies-Bouldin 指数 [12]、Silhouette 轮廓系数 [13] 可用于度量某些聚类目的下的结论性能。

3.2 算法概述与 K 值选择

注：本小节所使用算法及评价指标均采用 scikit_learn[14] 实现。

3.2.1 原型聚类：K-Means++

K-Means 是一种简单、高效的聚类算法，假设聚类结构能通过一组“原型”刻画，算法的主要思想是通过迭代过程把数据集划分为不同的类别，流程如图 8。K-means++ 优化“初始化 K 个聚类中心”，要求初始的聚类中心之间的相互距离要尽可能的远，在“孤立点数据敏感性”方面优于 K-Means 算法。默认采用欧式距离、重心法进行相似度量。

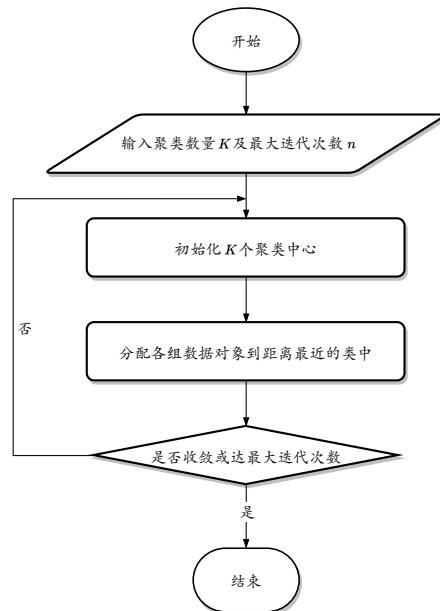


图 8 KMeans 算法流程图

将最大迭代次数设置为 1000，选择 K 等于 2~50 绘制聚合系数与卡林斯基-哈拉巴斯指数折线图。根据图 9，K 值从 2 到 13 时，畸变程度变化最大；超过 6 畸变程度变化显著降低：因此根据肘部法则，可将聚类数量 K 设定为 5；从来看，应将聚类数量设定为 6 以下。该结论符合卡林斯基-哈拉巴斯指数峰值，故将聚合数目设定为 5。

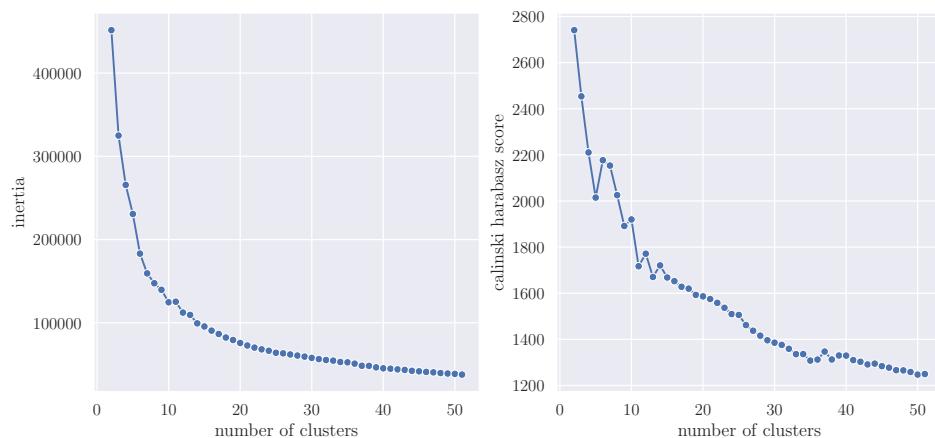


图 9 K-Means 聚合系数与卡林斯基-哈拉巴斯指数

3.2.2 层次聚类: BIRCH+AGNES

AGNES 算法 (Agglomerative Nesting)，以自底向上方式，不断重复合并，产生不同粒度（层次）的聚类结果，一般最终预设聚类数目为 1。该算法默认采用“欧式距离”进行度量样本距离，采用“离差平方和”(ward linkage) 作为簇距离度量函数，可通过聚类谱系图 (dendrogram) 可视化，算法执行流程如下：

Algorithm 1: AGNES 算法

input : 样本集 $D = \{x_1, x_2, \dots, x_m\}$;
 聚类簇距离度量函数 d ;
 聚类簇数 k 。

output: 簇划分: $\mathcal{C} = \{C_1, C_2, \dots, C_k\}$

```

1 # 先将每个样本视作一个初始簇构造;
2 # 构造  $M$  个类，每个类仅包含一个样本;
3 for  $j = 1, 2, \dots, m$  do
4    $C_j = x_j$ 
5 # 两两计算距离;
6 for  $i = 1, 2, \dots, m$  do
7   for  $j = i + 1, \dots, m$  do
8      $M_{i,j} = d(C_i, C_j)$ ;
9      $M_{j,i} = M_{i,j}$ 
10 # 当前类个数大于预设簇数;
11 while  $q > k$  do
12   合并距离最近的两个聚类簇  $C_{i^*} = C_{i^*} \cup C_{j^*}$ ;
13   for  $j = j^* + 1, j = j^* + 2, \dots, q$  do
14     将聚类簇  $C_j$  重编号为  $C_{j-1}$ 
15   删除距离矩阵  $M$  的第  $j^*$  行与第  $j^*$  列;
16   # 重新计算距离矩阵;
17   for  $j = 1, 2, \dots, q - 1$  do
18      $M_{i^*,j} = d(C_{i^*}, C_j)$ ;
19      $M_{j,i^*} = M_{i^*,j}$ 
20   q=q-1

```

AGNES 算法时间复杂度为立方级别，对于本题万级别用户而言，耗费时间过长。因

此，本文首先采用 BIRCH[15] 聚类算法得到初步聚类质心，作为 AGNES 算法的输入。BIRCH 算法综合了层次凝聚和迭代的重定位方法首先用自底向上的层次算法，然后用迭代的重定位来改进结果。它的主要思想是：扫描数据库，建立一个初始存放于内存中的聚类特征树然后对聚类特征树的叶结点进行聚类 [16]。

针对本题，首先将前 7 日 44253 名用户采用 BIRCH 算法聚类获得 2827 枚质心，而后 AGNES 对质心进行聚类，并绘制 6 层谱系图如下。

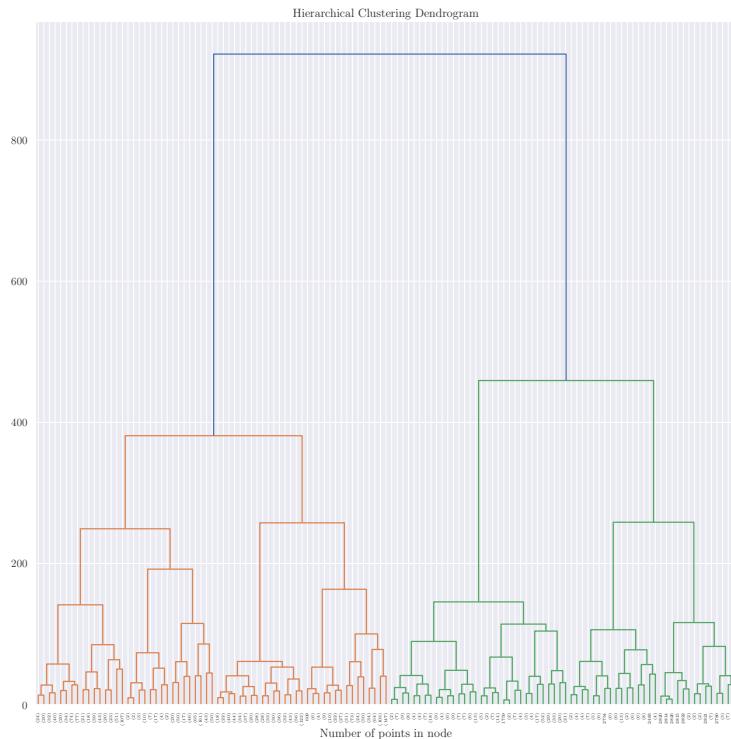


图 10 BIRCH+AGNES 谱系图

3.2.3 密度聚类：DBSCAN

DBSCAN 算法从样本密度的角度来考察样本之间的可连接性，要求聚类空间中的以 eps 为半径的邻域内所包含对象的数目不小于某一给定阈值 min_samples ，并基于可连接样本不断扩展生长聚类簇以获得最终的聚类结果 [1]。算法将数据样本点分为三类：

1. 核心点：在半径 eps 内含有不少于 min_samples 数目的点
2. 边界点：在半径 eps 内点的数量小于 min_samples ，但是落在核心点的邻域内
3. 噪声点：既不是核心点也不是边界点的点

DBSCAN 不需要预先输入要划分的聚类个数，但是对 eps 、 min_samples 参数敏感。记特征维度数目 $N = 3$ 、 $K = 2N - 1$ ，按照以下经验值确定超参数 [17, 18]: $\text{min_samples} = 2N = 6$ ，将数据集各点与 K-最近邻算法分类标签的距离排序，观察图 11 拐点 y 坐标确定 $\text{eps} = 2$ 。

运行结果如下，聚类数量为 7，噪点用户 539 名。

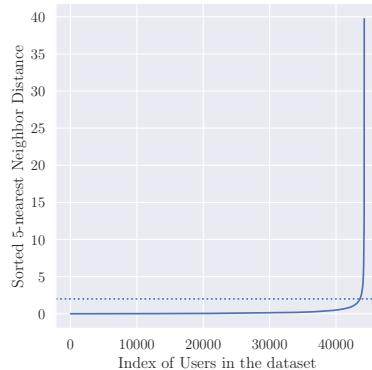


图 11 数据集各点 6-最近邻距离（排序）

```

• ~ from sklearn import metrics
  from sklearn.cluster import DBSCAN

db = DBSCAN(eps=2, min_samples=6).fit(X)
labels = db.labels_

# Number of clusters in labels, ignoring noise if present.
n_clusters_ = len(set(labels)) - (1 if -1 in labels else 0)
n_noise_ = list(labels).count(-1)

print(f"Estimated number of clusters: {n_clusters_}")
print(f"Estimated number of noise points: {n_noise_}")

✓ 0.0s
Estimated number of clusters: 7
Estimated number of noise points: 575
  
```

图 12 Visual Studio 运行结果截图

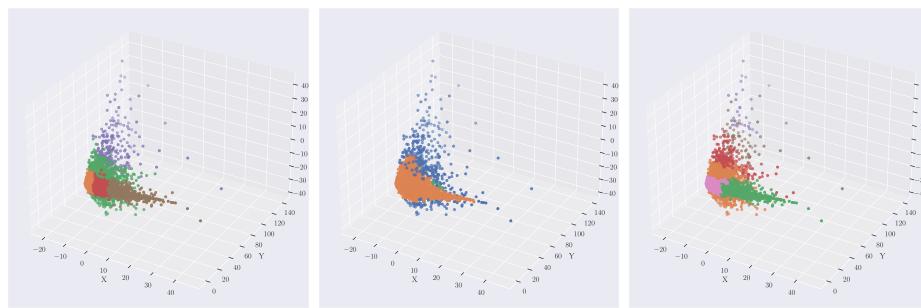
3.3 算法比较与用户画像

3.3.1 算法比较

截取 BIRCH+AGNES 聚类数目为 6 时的聚类结果。先采用内部评价指标比较，再使用 PCA 降维特征绘制三维散点图，最后使用二维 t-SNE[19] 可视化原始 80 维特征。

表 7 聚类算法内部评价指标

<i>Method</i>	<i>Calinski – Harabasz</i>	<i>Davies – Bouldin</i>	<i>Silhouette</i>
0 K-Means++	32187.076682	0.766536	0.511919
1 DBSCAN	2870.580217	2.817892	0.664019
2 BIRCH+AGNES	24732.933816	0.860360	0.622801



(a) K-Means++

(b) DBSCAN

(c) BIRCH+AGNES

图 13 PCA 三维散点图

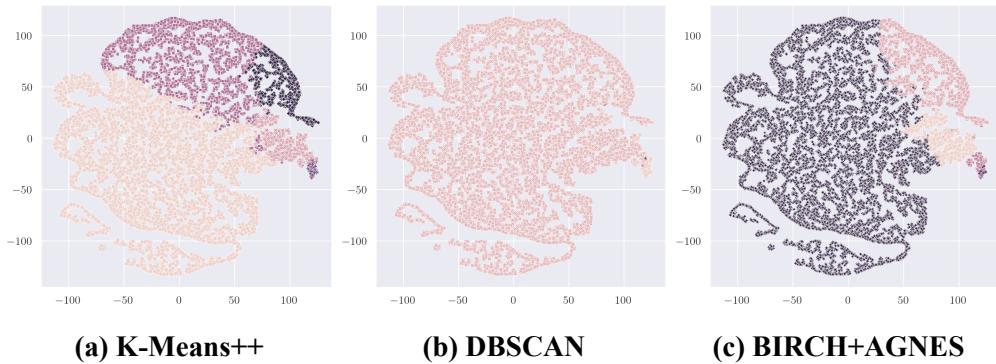


图 14 t-SNE 二维散点图

散点图显示 DBSCAN 聚类较为笼统, K-Means++ 与 BIRCH+AGNES 类间区隔更清晰; 从表 7 看, K-Means++ 更优, 故根据 K-Means++ 聚类结果对各用户群体画像。

3.3.2 用户画像

根据聚类结果进行数据可视化, 可以获得 5 号人群对 20 种 APP 使用时长排序:

- A 类 APP: 2 号 \approx 0 号 \geq 1 号 \approx 4 号 \geq 3 号
- B 类 APP: 2 号 \approx 0 号 \geq 4 号 \geq 1 号 \geq 3 号
- C 类 APP: 3 号 \approx 1 号 \geq 2 号 \approx 0 号 \geq 4 号
- D 类 APP: 0 号 \geq 2 号 \geq 1 号 \approx 4 号 \geq 3 号
- E 类 APP: 0 号 \geq 2 号 \geq 1 号 \approx 4 号 \geq 3 号
- F 类 APP: 4 号 \geq 2 号 \geq 1 号 \approx 3 号 \geq 0 号
- G 类 APP: 2 号 \geq 0 号 \geq 1 号 \approx 4 号 \geq 3 号
- H 类 APP: 2 号 \geq 0 号 \geq 1 号 \approx 4 号 \geq 3 号
- I 类 APP: 2 号 \geq 0 号 \geq 4 号 \approx 1 号 \geq 3 号
- J 类 APP: 2 号 \geq 0 号 \geq 4 号 \approx 1 号 \geq 3 号
- K 类 APP: 2 号 \approx 0 号 \approx 4 号 \approx 1 号 \geq 3 号
- L 类 APP: 2 号 \approx 0 号 \approx 4 号 \approx 1 号 \approx 3 号
- M 类 APP: 2 号 \approx 0 号 \geq 4 号 \approx 1 号 \approx 3 号
- N 类 APP: 2 号 \approx 0 号 \geq 4 号 \approx 1 号 \geq 3 号
- O 类 APP: 2 号 \approx 0 号 \approx 4 号 \approx 1 号 \geq 3 号
- P 类 APP: 2 号 \approx 4 号 \geq 0 号 \approx 1 号 \geq 3 号
- Q 类 APP: 2 号 \approx 0 号 \geq 4 号 \approx 1 号 \geq 3 号
- R 类 APP: 2 号 \approx 0 号 \geq 4 号 \approx 1 号 \geq 3 号
- S 类 APP: 2 号 \approx 0 号 \approx 4 号 \geq 1 号 \geq 3 号
- T 类 APP: 2 号 \approx 0 号 \geq 4 号 \approx 1 号 \geq 3 号

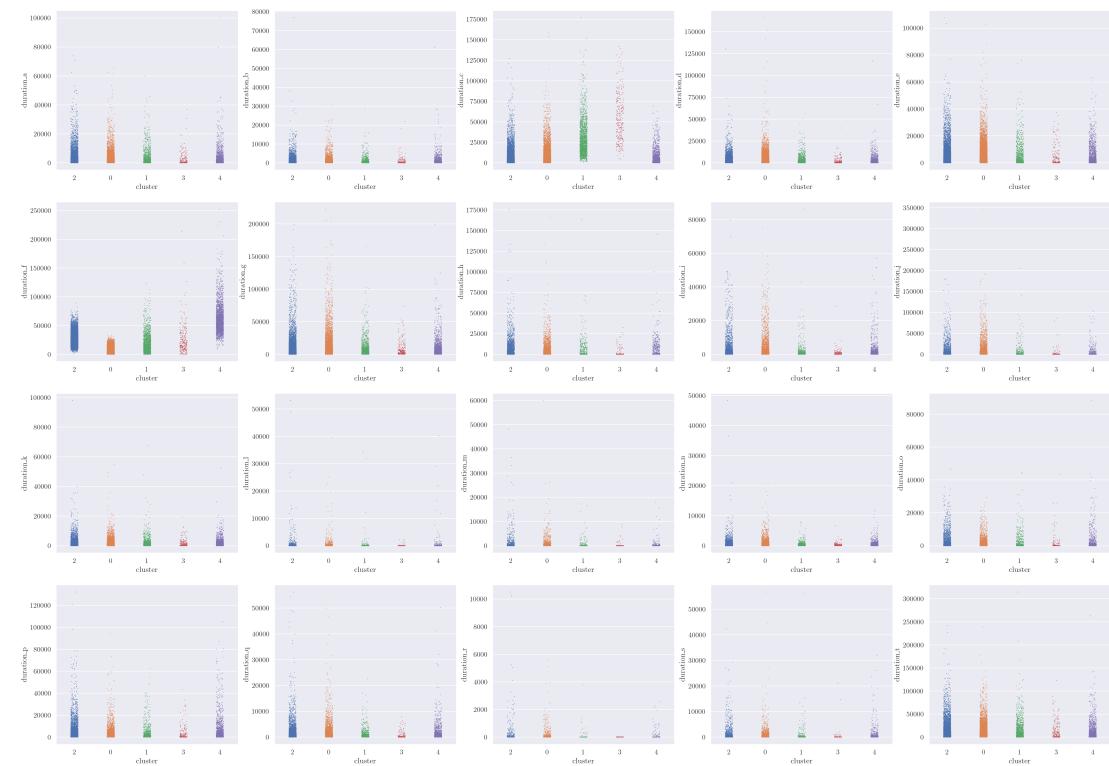


图 15 使用时长

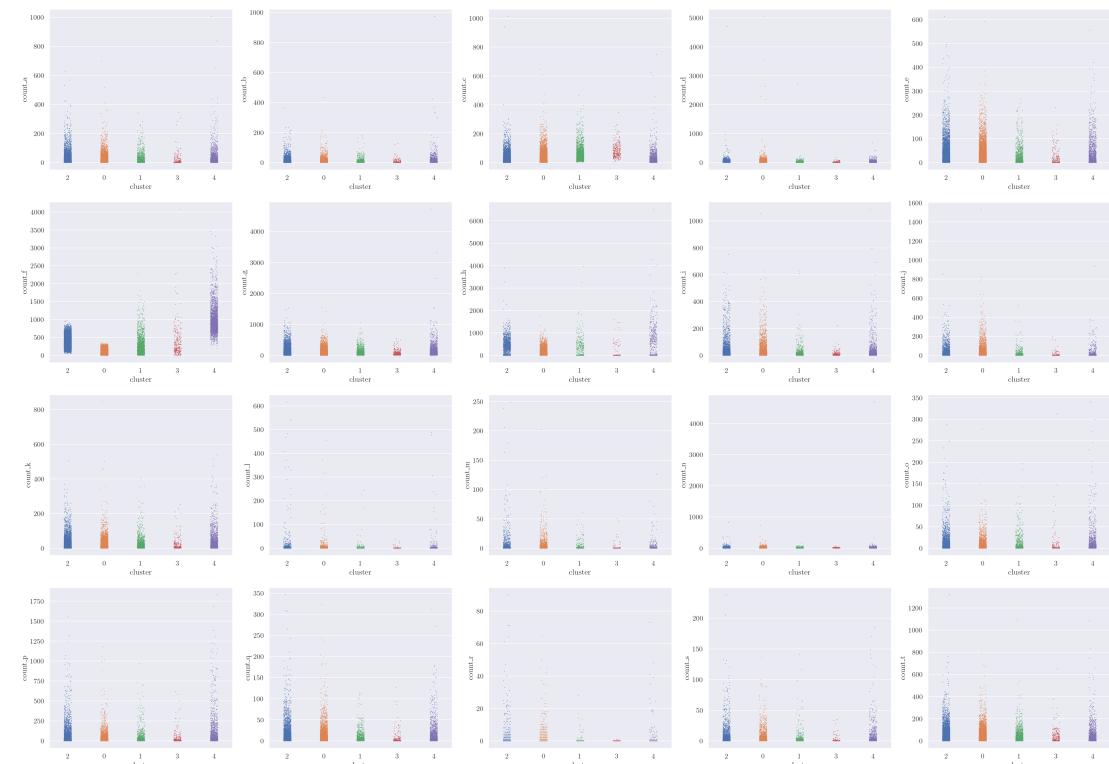


图 16 使用频次

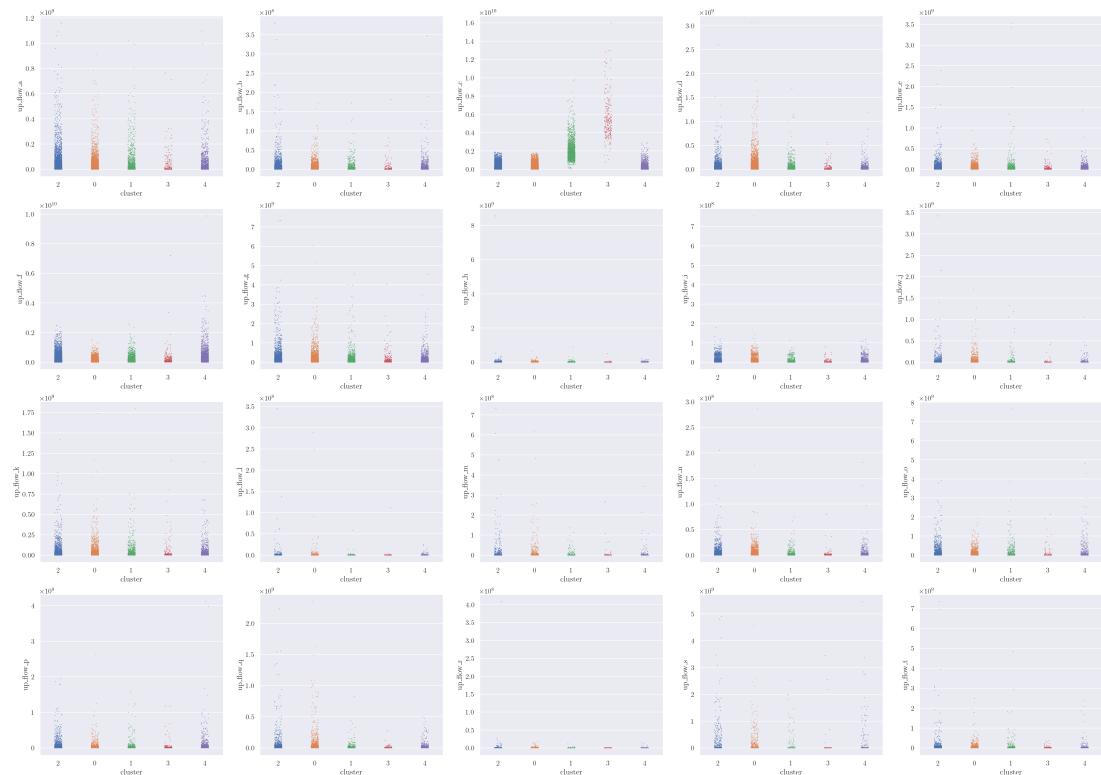


图 17 上行流量

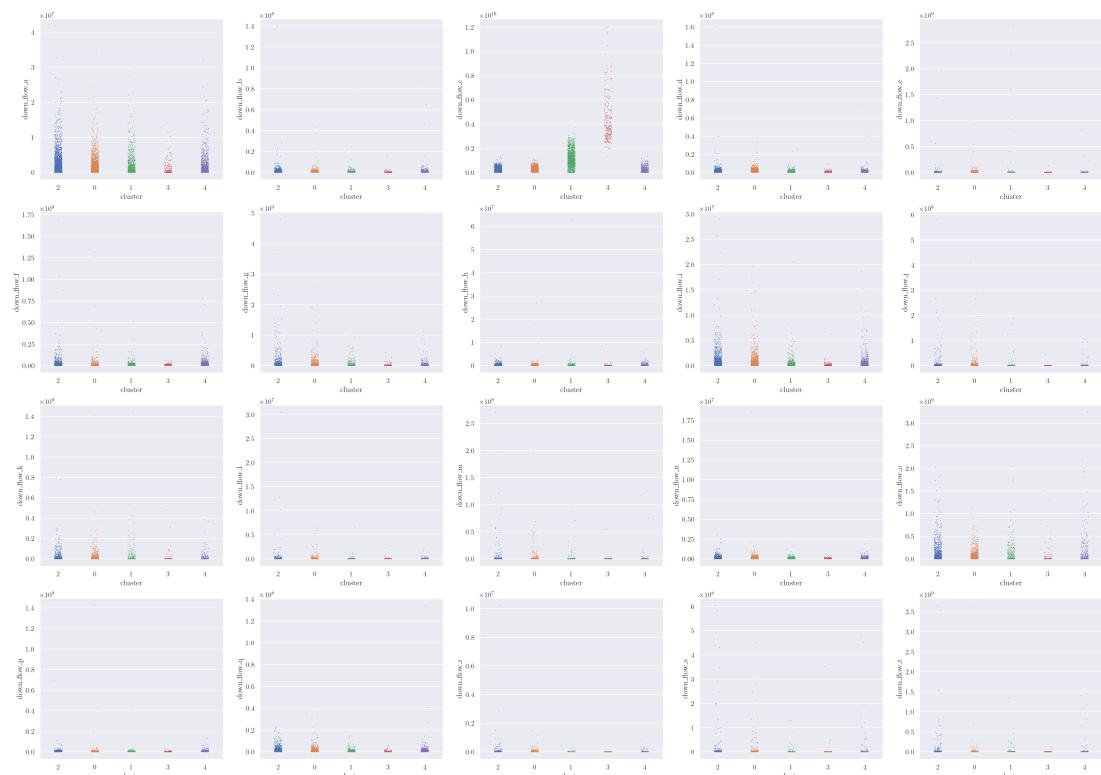


图 18 下行流量

就图 8 总体情况而言，4 号用户群使用智能手机的频次最高，使用时长较高，可能需要时刻刷新手机；0 号用户群使用智能手机的频次最低，时长最短，对手机依赖程度不强；1 号、3 号用户群使用流量最多；相对于 4 号而言，3 号用户群不经常刷新手机，单次使用 APP 时间较长。

表 8 聚类各簇用户统计描述

cluster	count_sum	duration_sum	up_flow_sum	down_flow_sum
0	375.776880	43624	392653844	65333138
1	1177.867654	159571	5290795162	2128662377
2	1342.008467	122244	1131347214	167123031
3	1573.553903	232744	12630933652	8764644593
4	2931.083175	196532	1621400073	235777958

就图 19 各类 APP 情况而言，1 号、3 号用户群似乎对 C 类 APP“情有独钟”；F 类 APP 使用情况最为参差，4 号使用时长最长，降序依次是 2 号、3 号、1 号、0 号。

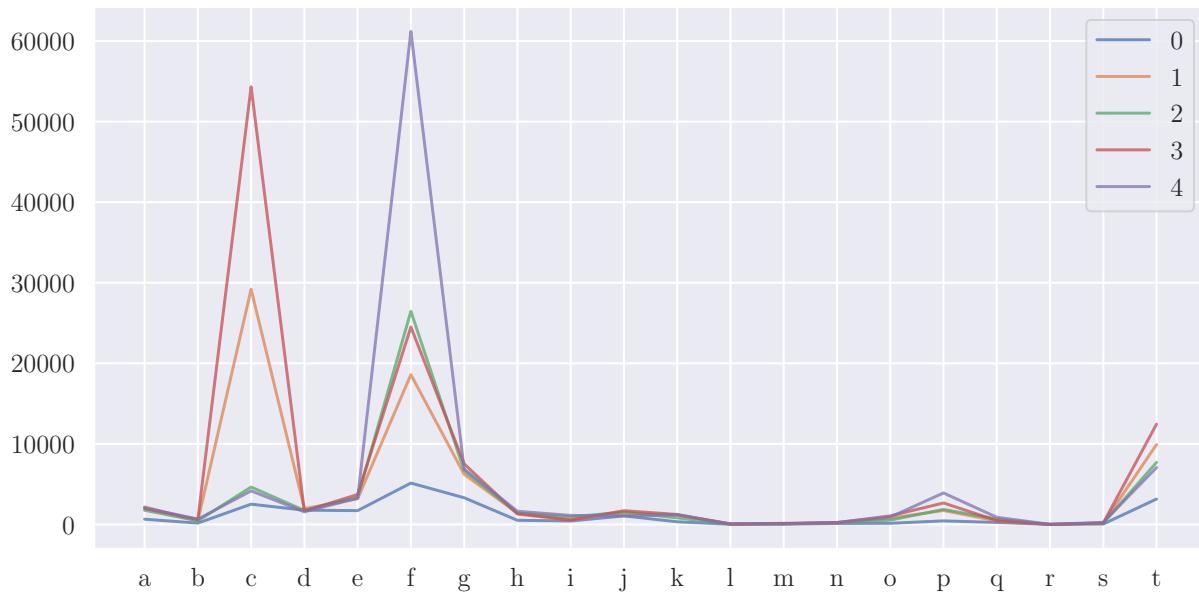


图 19 各类用户群对各类 APP 的使用时长（均值）

四、问题二：未来使用情况预测

4.1 评价指标与求解思路

4.1.1 评价指标

对于二元分类任务（第一小问）而言，实际及预测标签输出均为二值变量（真或假，1 或 0）时，评价指标一般采用混淆矩阵（图 20）定义：

		Ground Truth		Predictive Label
		True Positive	False Positive	
Ground Truth	False Negative	True Negative		

图 20 混淆矩阵

准确率（accuracy）表示正确预测的比率，表示预测正确的数据占所有资料的比率：

$$acc = \frac{TP + TN}{TP + TN + FP + FN} \quad (3)$$

此外，精准率（precision）、召回率（recall）、调和平均 F1 值也经常作为评价指标：

$$prec = \frac{TP}{TP + FP} \quad (4)$$

$$recall = \frac{TP}{TP + FN} \quad (5)$$

$$F_1 = \frac{2}{\frac{1}{prec} + \frac{1}{recall}} \quad (6)$$

对于回归任务（第二小问）而言，常用评价指标包括均方根误差（RMSE）、均方根对数误差（RMSLE）、平均绝对值误差（MAE）、决定系数（ R^2 ）等，本题采用 NMSE：

$$RMSE = \sqrt{\frac{1}{N} \sum_{i=1}^N (y_i - \hat{y}_i)^2} \quad (7)$$

$$RMSLE = \sqrt{\frac{1}{N} \sum_{i=1}^N (\log(1 + y_i) - \log(1 + \hat{y}_i))^2} \quad (8)$$

$$MAE = \frac{1}{N} \sum_{i=1}^N |y_i - \hat{y}_i| \quad (9)$$

$$R^2 = 1 - \frac{\sum_{i=1}^N (y_i - \hat{y}_i)^2}{\sum_{i=1}^N (y_i - \bar{y}_i)^2} \quad (10)$$

$$\bar{y} = \frac{1}{N} \sum_{i=1}^N y_i \quad (11)$$

$$NMSE = \sqrt{\frac{\sum_{i=1}^N (y_i - \hat{y}_i)^2}{\sum_{i=1}^N (y_i - \bar{y}_i)^2}} \quad (12)$$

公式中符号意义如下：

表 9 评价指标符号意义

符号	意义
N	资料笔数，本题代表 1~21 天出现的用户总数
i	$1, 2, \dots, N$ ，为每笔资料的索引
y_i	第 i 笔资料的实际值，如第 i 名用户第 12 至 21 天使用 a 类 APP 的实际有效日均使用时长
\hat{y}_i	第 i 笔资料的预测值，如第 i 名用户第 12 至 21 天使用 a 类 APP 的预测有效日均使用时长

4.1.2 系统 Pipeline

在使用机器学习建立模型解决第一小问与第二小问前，必须先确定模型输入与输出的固定格式与含义。从系统本身而言，问题二要求输入 1~11 天用户使用 APP 监测数据，输出 12~21 天用户是否使用 A 类 APP 以及日均使用时长。这一问题兼具“用户行为属性”与“时间序列属性”，我们可以将这一复杂问题在两个方向进行抽象、简化：

1. 输入某用户前一天是否使用 A 类 APP（类别），输出该用户后 1 天是否使用（类别）
2. 输入某用户前 n 天是否使用 A 类 APP（向量），输出该用户后 1 天是否使用（类别）
3. 输入某用户前 n 天是否使用 A 类 APP（向量），输出该用户后 n 天是否使用（向量）
4. 输入某用户前 1 天的使用时长（标量），输出一个用户后 1 天的使用时长（标量）
5. 输入某用户前 n 天的使用时长（向量），输出一个用户后 1 天的使用时长（标量）
6. 输入某用户前 n 天的使用时长（向量），输出一个用户后 n 天的使用时长（向量）

由于 12~21 天属于测试集，不可泄漏参与训练，因此天然缺失构建 $n = 10$ 的 6 号、3 号求解框架，如果选取 $n = 3$ 或许不符合以 7 天为周期的社会性活动常识。因此，本文折衷资料数目与特征数目，提出了两种求解框架进行模式识别、特征挖掘：

1. 输入某用户前 n 天的使用时长（向量），输出一个用户后 1 天的使用时长（标量），并通过预训练模型执行下游任务：滑动窗口覆盖 21 天，如图 21。
2. 输入某用户前 n 天的使用时长（向量），输出一个用户后 m 天的使用时长（向量），并通过预训练模型执行下游任务：滑动窗口覆盖 21 天，如图 22。

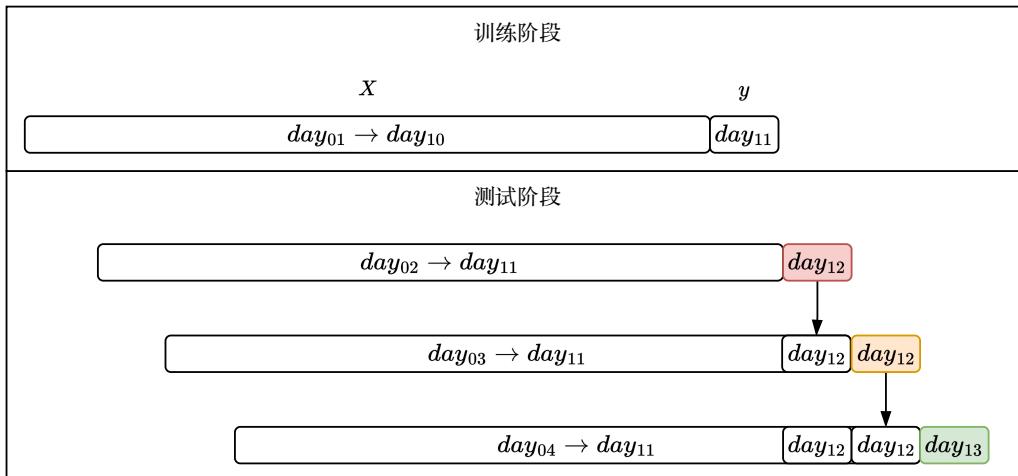


图 21 求解框架（一）

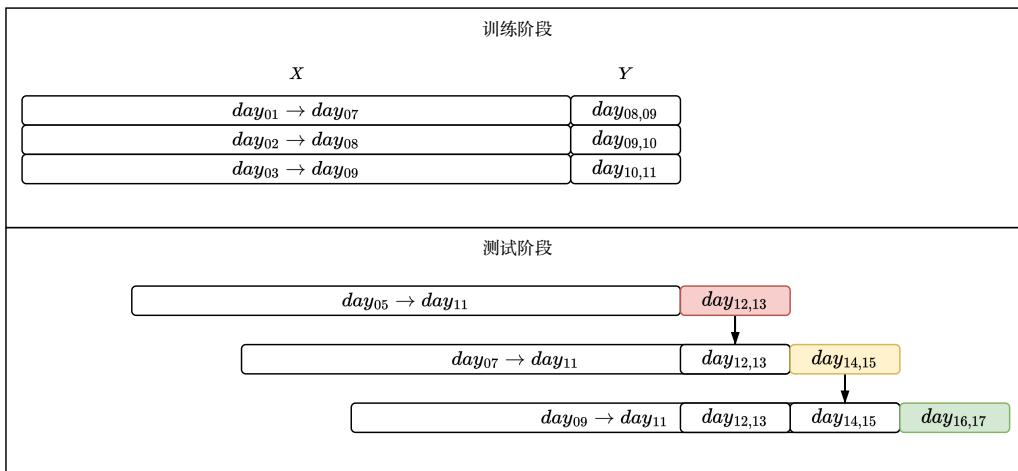


图 22 求解框架（二）

对于提出的两种求解框架，考虑到实现细节与篇幅页数的限制，下文仅讨论和展示框架（一），根据监测数据每日 A 类 APP 有效使用时长作为特征建立训练集、测试集，共计 47345 笔资料。

4.2 超参数选择与预训练模型建立

4.2.1 建立 KNN 最近邻分类预测模型

KNN 将无类别标签样本归为最接近该点的 K 个样本点中最频繁使用的一类。本文在 1~40 所有奇数中，对 K 值进行网格搜索，在训练资料上对模型采用 8 折交叉验证，运行结果显示： $K = 33$ 时，正确率最高可达 87.508%。

4.2.2 建立 LogisticRegression 分类预测模型

LogisticRegression 也称“对数概率回归”，属于一种基于概率的判别模型。在训练资料上对模型采用 8 折交叉验证，运行结果显示：模型在预训练资料上正确率平均为 87.31%。

4.2.3 建立随机森林集成分类预测模型

随机森林是一个集成装箱多个决策树的分类器，每棵树模型采用 Bootstrap 方式对特征进行采样。本文对表 10 三类超参数进行网格搜索，在训练资料上对模型采用 3 折交叉验证，运行结果显示，设置下列超参数时，正确率最高可达 88.71%。

表 10 随机森林最佳超参数

超参数	取值	超参数	取值	超参数	取值
$n_estimators$	60	$min_samples_leaf$	7	max_depth	12

4.2.4 建立 XGBoost 集成回归预测模型

XGBoost 也是以决策树为基础的模型，采用梯度提升对多棵决策树进行组合，在准确率、运算速度、操作便利性上表现优秀。采用网格搜索超参数，在训练资料上对模型采用 10 折交叉验证，选择超参数如表 11 时，在预训练资料上（给定 1~10 天，预测第 11 天）NMSE 损失最小可达 0.8262。

表 11 xgboost 最佳超参数

超参数	取值	超参数	取值	超参数	取值
$n_estimators$	10	max_depth	3	min_child_weight	1.0

4.3 运行结果

4.3.1 运行结果

在选择最优的预训练模型上，经测试，在下游任务 1“预测第 12 21 天是否会使用该类 APP”上随机森林算法正确率高达；在下游任务 2“预测第 12 21 天用户使用 a 类 APP 的有效日均使用时长”中 XGboost 的 NMSE 低至。

参考文献

- [1] 周志华. 机器学习 [M]. 北京. 清华大学出版社. 2016. 197-219
- [2] 何宏. 高维数据的聚类分析 [M]. 上海. 上海交通大学出版社. 2022. 1-16
- [3] 陈志泊, 韩慧, 王建新, 孙俏, 聂耿青. 数据仓库与数据挖掘 [M]. 北京. 清华大学出版社. 2009
- [4] 常乐. 基于用户行为分析的用户画像系统设计与实现 [D]. 北京邮电大学. 2020
- [5] 陈纯, 龙瀛, 黄贵恺. 屏幕使用时间与步行活动关系的探索性研究 [J]. 景观设计学 (中英文). 2021. 9(04):68-81
- [6] 成雪, 于冬梅, 赵丽云等. 2016—2017 年中国各省中小学生电子屏幕使用现状 [J]. 卫生研究. 2023,52(03):382-387
- [7] 武慧娟, 赵天慧, 孙鸿飞等. 基于支付意愿的数字阅读用户画像聚类研究 [J]. 情报科学. 2022. 40(05)
- [8] 侯金凤. 移动互联网下手机用户使用行为特征的研究 [J]. 电脑知识与技术. 2016,12(07)
- [9] 韦磊. 基于移动终端数据的用户画像模型研究 [D]. 江苏科技大学. 2021
- [10] Garg, R., & Barpanda, S. Machine Learning Algorithms for Time Series Analysis and Forecasting[Z/OL]. arXiv preprint arXiv:2211.14387. <https://arxiv.org/abs/2211.14387>
- [11] T. Caliński, J Harabasz. A dendrite method for cluster analysis[J]. Communications in Statistics. 1974. 3:1, 1-27
- [12] Davies D L , Bouldin D W. A Cluster Separation Measure[J]. IEEE Transactions on Pattern Analysis and Machine Intelligence. 1979. PAMI-1(2):224-227
- [13] Peter R J . Silhouettes: A graphical aid to the interpretation and validation of cluster analysis[J]. Journal of Computational & Applied Mathematics. 1987. 20
- [14] Swami A , Jain R. Scikit-learn: Machine Learning in Python[J]. Journal of Machine Learning Research. 2013, 12(10):2825-2830
- [15] Zhang T , Ramakrishnan R. Miron Livny: BIRCH: An Efficient Data Clustering Method for Very Large Databases[J]. ACM SIGMOD Record. 1999. 25(2)

- [16] 赵玉艳, 郭景峰, 郑丽珍等. 一种改进的 BIRCH 分层聚类算法 [J]. 计算机科学. 2008(03):180-182+208
- [17] Sander J , Ester M , Kriegel H P ,et al. Density-Based Clustering in Spatial Databases: The Algorithm GDBSCAN and Its Applications[J]. Data Mining & Knowledge Discovery. 1998. 2(2):169-194
- [18] Schubert E , Sander J , Ester M ,et al. DBSCAN revisited, revisited: Why and how you should (still) use DBSCAN[J]. ACM Transactions on Database Systems, 2017, 42(3):1-21
- [19] Maaten L J P V D , Hinton G E. Visualizing High-Dimensional Data using t-SNE[J]. Journal of Machine Learning Research. 2008. 9:2579-2605.

附录 A 运行环境与环境依赖

代码在 Python3.9.12, MacBook Air Apple M1 MacOS13.0.1 (22A400) 测试无误。下载解压初赛数据集，将 app_class.csv、dayxx.txt 放至 Datasets 文件夹中，在 Visual Studio Code 或 Terminal 中运行 ‘preprocessing.py‘ 的 ‘generate_pickle_origin‘，获得 pickle 格式的粗处理数据，对应 2.1、2.2、2.3、2.4 小节。

除数据清洗、预处理源代码直接来自 Python 脚本，其余附录所列出的代码均由 Jupyter Notebook 记事本文件导出。

```
matplotlib~=3.7.2
numpy~=1.23.5
pandas~=2.0.3
scienceplots~=2.1.0
scikit_learn~=1.3.0
seaborn~=0.12.2
tqdm~=4.64.0
xgboost~=1.7.6
ydata_profiling~=4.3.1
```

附录 B 数据清洗、预处理源代码

```
from tqdm.auto import tqdm
import pandas as pd
import os

def generate_pickle_origin(start_day=1, end_day=21):
    # 读取类型数据
    app = pd.read_csv(
        filepath_or_buffer='..../Datasets/app_class.csv',
        header=None
    ).drop_duplicates()

    # 更新列标签
    app.columns = ['appid', 'app_class']
```

```
# 新增一笔虚拟资料 (NaN)
app = pd.concat(
    objs=[pd.DataFrame({'appid': [-404], "app_class": ['NaN']}), app],
    ignore_index=True
)

# 约束数据格式
app['app_class'] = app['app_class'].astype('category')
app['appid'] = app['appid'].astype('category')

# 初赛数据集day1~21
for i in tqdm(range(start_day, end_day + 1)):
    # 读取监测数据
    df = pd.read_csv(
        filepath_or_buffer=f'../Datasets/day{str(i).zfill(2)}.txt',
        header=None
    ).drop_duplicates()

    # 更新列标签
    df.columns = ['uid', 'appid', 'app_type', 'start_day', 'start_time',
                  'end_day', 'end_time', 'duration', 'up_flow', 'down_flow']

    # 新增app_class, 对于所属类别未知的APP, 类别记作NaN
    df = df.merge(app, on='appid', how='left')
    df['app_class'] = df['app_class'].fillna('NaN')

    # 约束数据格式
    df['appid'] = df['appid'].astype('category')
    df['uid'] = df['uid'].astype('category')
    df['app_type'] = df['app_type'].astype('category')

    # df['start_time'] = pd.to_datetime(df['start_time'], format="%H:%M:%S")
    # df['end_time'] = pd.to_datetime(df['end_time'], format="%H:%M:%S")
    # 使用 pandas.Timestamp 将 `day` 和 `time` 合并
    # df['start_time_new'] = df.apply(
    #     lambda x: x['start_time'] + pd.Timedelta(x['start_day'] - 1,
    #                                              unit='D'), axis=1)
```

```
# df['end_time_new'] = df.apply(
#     lambda x: x['end_time'] + pd.Timedelta(x['end_day'] - 1,
#     unit='D'), axis=1)

df = df[['uid', 'appid', 'app_type', 'app_class', 'start_day',
'start_time',
'end_day', 'end_time', 'duration', 'up_flow', 'down_flow']]

# 使用 pickle 存储
df.to_pickle(f'../Datasets/day{str(i).zfill(2)}.pkl')

def generate_pickle_1(start_day=1, end_day=21):
    """
    数据融合: day01~day21
    建立特征: 连接辅助表, 新增app_class
    清洗数据: 异常值、缺失值、重复值
    """

    # 读取类型数据
    app = pd.read_csv(
        filepath_or_buffer='../Datasets/app_class.csv',
        header=None
    ).drop_duplicates()

    # 更新列标签
    app.columns = ['appid', 'app_class']

    # 新增一笔虚拟资料 (NaN)
    app = pd.concat(
        objs=[pd.DataFrame({'appid': [-404], "app_class": ['NaN']}), app],
        ignore_index=True
    )

    # 约束数据格式
    app['app_class'] = app['app_class'].astype('category')
    app['appid'] = app['appid'].astype('category')
```

```
# 初赛数据集day1~21
for i in tqdm(range(start_day, end_day + 1)):
    # 读取监测数据
    df = pd.read_csv(
        filepath_or_buffer=f'../Datasets/day{str(i).zfill(2)}.txt',
        header=None
    ).drop_duplicates()

    # 更新列标签
    df.columns = ['uid', 'appid', 'app_type', 'start_day', 'start_time',
                  'end_day', 'end_time', 'duration', 'up_flow', 'down_flow']

    # 问题: start_day为负数, 甚至持续时间长达一千年
    df = df.query('start_day >=0 & duration <= 9159')

    # 新增app_class, 对于所属类别未知的APP, 类别记作NaN
    df = df.merge(app, on='appid', how='left')
    df['app_class'] = df['app_class'].fillna('NaN')
    # 问题: app_type 列存在中文
    df['app_class'] = df['app_class'].replace({'用户': 'usr', '预装': 'sys'})

    # 约束数据格式
    df['appid'] = df['appid'].astype('category')
    df['uid'] = df['uid'].astype('category')
    df['app_type'] = df['app_type'].astype('category')

    # df['start_time'] = pd.to_datetime(df['start_time'], format="%H:%M:%S")
    # df['end_time'] = pd.to_datetime(df['end_time'], format="%H:%M:%S")
    # 使用 pandas.Timestamp 将 `day` 和 `time` 合并
    # df['start_time_new'] = df.apply(
    #     lambda x: x['start_time'] + pd.Timedelta(x['start_day'] - 1,
    #                                               unit='D'), axis=1)
    # df['end_time_new'] = df.apply(
    #     lambda x: x['end_time'] + pd.Timedelta(x['end_day'] - 1,
    #                                               unit='D'), axis=1)
```

```
df = df[['uid', 'appid', 'app_type', 'app_class', 'start_day',
         'start_time',
         'end_day', 'end_time', 'duration', 'up_flow', 'down_flow']]

# 使用 pickle 存储
df.to_pickle(f'../Datasets/day{str(i).zfill(2)}.pkl')
```

附录 C 数据探索：数据概览源代码

```
#!/usr/bin/env python
# coding: utf-8

# # 载入套件


from typing import Dict
from tqdm.auto import tqdm

import numpy as np
import random
import pandas as pd
import ydata_profiling

import scienceplots
import matplotlib.pyplot as plt
import seaborn as sns
import matplotlib

import torch
import os

get_ipython().run_line_magic('matplotlib', 'inline')

pd.plotting.register_matplotlib_converters()
sns.set_style("whitegrid")
```

```
sns.set_palette("RdBu")
sns.set(
    rc={'text.usetex': True},
    font="serif",
    font_scale=1.2
)

days = {i: pd.read_pickle(f'../Datasets/day{str(i).zfill(2)}.pkl')
        for i in range(1, 22)}

# # 工具函数

SEED = 20230723

def same_seed(seed=SEED):
    random.seed(seed)
    os.environ['PYTHONHASHSEED'] = str(seed)
    np.random.seed(seed)
    torch.manual_seed(seed)
    torch.cuda.manual_seed(seed)
    torch.backends.cudnn.deterministic = True
    torch.backends.cudnn.benchmark = False
    if torch.cuda.is_available():
        torch.cuda.manual_seed_all(seed)

same_seed()

# # 辅助表格

# 辅助表格，常用APP类别
# 发现：原始表格存在重复值
app = pd.read_csv('../Datasets/app_class.csv', header=None).drop_duplicates()
```

```
app.columns = ['appid', 'app_class']

# 副本：不锁定数据格式
app_copy = app.copy()

# 约束数据格式
app['appid'] = app['appid'].astype('category')
app['app_class'] = app['app_class'].astype('category')

# 打印行列数、概况
print('app:', app.shape)
pd.DataFrame(app.value_counts('app_class'))
app.describe()

# 21天内监测数据中共有36435种app_id
appid = set({})
for i in range(1, 22):
    appid = appid.union(days[i]['appid'].unique())
len(appid)

# NaN
val = pd.DataFrame({'appid': list(appid)})
val = val.merge(app_copy, on='appid', how='left')
val[val['app_class'].isna()].shape

# a~t
val['app_class'].dropna().shape

sns.catplot(kind='count', data=val.sort_values(by='app_class'),
```

```
x='app_class', height=3.5, aspect=10/3.5)

plt.savefig("../Thesis/figures/app_class_countplot_in_days.pdf",
           dpi=400, bbox_inches='tight', pad_inches=0)

pd.DataFrame(val.fillna('NaN').value_counts(
    ['app_class'])).sort_values(by='count')
```

附录 D 数据探索：类别变量代码

```
#!/usr/bin/env python
# coding: utf-8

# # 载入套件

# In[1]:


from typing import Dict
from tqdm.auto import tqdm

import numpy as np
import random
import pandas as pd
import ydata_profiling

import scienceplots
import matplotlib.pyplot as plt
import seaborn as sns
import matplotlib

import torch
import os
```

```
get_ipython().run_line_magic('matplotlib', 'inline')

pd.plotting.register_matplotlib_converters()
sns.set_style("whitegrid")
sns.set_palette("RdBu")
sns.set(
    rc={'text.usetex': True},
    font="serif",
    font_scale=1.2
)

days = {i: pd.read_pickle(f'../Datasets/day{str(i).zfill(2)}.pkl')
        for i in range(1, 22)}

# # 监测表格
#
# | 变量名 | 释义 |
# |-----|-----|
# | uid | 用户的id |
# | appid | APP的id (与app_class文件中的第一列对应) |
# | app_type | APP类型: 系统自带、用户安装 |
# | start_day | 使用起始天, 取值1-30 |
# | start_time | 使用起始时间 |
# | end_day | 使用结束天 |
# | end_time | 使用结束时间 |
# | duration | 使用时长 (秒) |
# | up_flow | 上行流量 |
# | down_flow | 下行流量 |

# # 单一变量

# In[2]:
```

发现: 数据集存在中文['用户', '预装']

```
temp = days[2].select_dtypes('category').describe()
```

```
temp

# In[3]:


app_type = set({})
for i in range(1, 22):
    app_type = app_type.union(days[i]['app_type'].unique())
app_type

# In[4]:


for i in range(1, 22):
    days[i].replace({'用户': 'usr', '预装': 'sys'}, inplace=True)

# # 天数变化

# In[5]:


data = {'day': [], 'cat': [], 'val': []}
for i in range(1, 22):
    temp = days[i].select_dtypes('category').describe()
    # 日活跃用户
    data['day'].append(i)
    data['cat'].append('users')
    data['val'].append(temp.loc['unique', 'uid'])

    # 日志
    data['day'].append(i)
    data['cat'].append('logs')
    data['val'].append(temp.loc['count', 'uid'])
```

```
# APP
data['day'].append(i)
data['cat'].append('apps')
data['val'].append(temp.loc['unique', 'appid'])
temp = pd.DataFrame(data)

# In[6]:
for cat in ['logs', 'users', 'apps']:
    g = sns.relplot(kind='line',
                      data=temp[temp['cat'] == cat],
                      x='day',
                      y='val',
                      marker='o')
    g.set(
        ylabel=f'val_{cat}',
        xticks=range(1, 22)
    )
    plt.savefig(f'../Thesis/figures/relplot_line_day_val_{cat}.pdf',
                dpi=400,
                bbox_inches='tight',
                pad_inches=0)

# # 变量关联

# In[7]:
# dfs = []
# for i in range(1, 22):
#     dfs.append(days[i][['appid', 'app_type', 'app_class']].drop_duplicates())
# ((pd.concat(dfs, ignore_index=True)).drop_duplicates()
# ).to_pickle('../Datasets/app.pkl')
```

```
# In[8]:  
  
app = pd.read_pickle('../Datasets/app.pkl')  
app['appid'] = app['appid'].astype('category')  
app.select_dtypes('category').describe()  
  
# In[9]:  
  
app.groupby('appid').count().query('app_class == 2')  
  
# In[10]:  
  
days[1][(days[1]['appid'] == 4) & (days[1]['app_type'] == 'usr')  
        & (days[1]['uid'] == '91C16B1FE338DA085CB0B0840D8C6BA5')]  
  
# In[11]:  
  
g = sns.catplot(kind='count', data=app.query("app_class != 'NaN'"),  
                 x='app_class',  
                 hue='app_type', hue_order=['usr', 'sys'],  
                 height=5, aspect=10/5)  
plt.yscale('log')  
plt.savefig("../Thesis/figures/catplot_count_app_class.pdf",  
            dpi=400, bbox_inches='tight', pad_inches=0.005)
```

附录 E 数据探索：数值变量代码

```
#!/usr/bin/env python
```

```
# coding: utf-8

# # 载入套件

# In[1]:


from typing import Dict
from tqdm.auto import tqdm

import numpy as np
import random
import pandas as pd
import ydata_profiling

import scienceplots
import matplotlib.pyplot as plt
import seaborn as sns
import matplotlib

import torch
import os

get_ipython().run_line_magic('matplotlib', 'inline')

pd.plotting.register_matplotlib_converters()
sns.set_style("whitegrid")
sns.set_palette("RdBu")
sns.set(
    rc={'text.usetex': True},
    font="serif",
    font_scale=1.2
)

days = {i: pd.read_pickle(f'../Datasets/day{str(i).zfill(2)}.pkl')
        for i in range(1, 22)}
```

```
# 继承自2.2数据探索之类别变量
for i in range(1, 22):
    days[i]['app_type'] = days[i]['app_type'].replace(
        {'用户': 'usr', '预装': 'sys'})

# # 单一变量

# In[2]:


days[1].select_dtypes('number').describe()

# In[3]:


fig, axes = plt.subplots(1, 3, figsize=(13.5, 5))

for i, x in enumerate(['duration', 'up_flow', 'down_flow']):
    g = sns.boxenplot(data=days[1], y=x, orient='v', ax=axes[i])
    g.set(
        yscale='log'
    )

plt.savefig("../Thesis/figures/boxenplot_duration_day01.pdf",
           dpi=400, bbox_inches='tight', pad_inches=0.005)

# In[4]:


boss = 0
for i in range(1, 22):
    boss = max(boss, days[i]['duration'].quantile(0.9998))
boss
```

```
# In[5]:  
  
# days[1].query("uid == '64B3E40461C56847F35DB46D55707EA4'").sort_values(  
#     by='end_time')  
  
# In[6]:  
  
# days[1].query("uid == '1F1EF788E30A9EA8E8CB52059A4B02A0'").sort_values(  
#     by='end_time')  
  
# In[7]:  
  
days[19].query("app_class == 'a' & duration > 20000")  
  
# In[8]:  
  
# 发现：有‘僵尸用户’（请求/活跃程度极低，监控记录中app_class均为NaN）  
days[1][days[1]['uid'] == '3B0AAFB3213D6DB0CCB17EDEAE80C38']  
  
# In[9]:  
  
# 数据清洗  
for i in range(1, 22):  
    days[i] = days[i].query('duration <= 9159')  
  
# In[10]:
```

```
days[1].select_dtypes('number').describe()

# In[52]:


fig, axes = plt.subplots(1, 5, figsize=(28, 4))
ax = axes[0]
g = sns.histplot(
    data=days[1].pivot_table(
        index='uid',
        values='appid',
        aggfunc='count',
    ),
    ax=ax,
    x='appid'
)

ax = axes[1]
g = sns.histplot(
    data=days[1].pivot_table(
        index='uid',
        values='duration',
        aggfunc=np.sum
    ),
    ax=ax,
    x='duration'
)

ax = axes[2]
g = sns.histplot(
    data=days[1].pivot_table(
        index='uid',
```

```
    values='up_flow',
    aggfunc=np.sum
),
ax=ax,
x='up_flow',
)
g.set(
    xscale='log'
)

ax = axes[3]
g = sns.histplot(
    data=days[1].pivot_table(
        index='uid',
        values='down_flow',
        aggfunc=np.sum
),
ax=ax,
x='down_flow',
)
g.set(
    xscale='log'
)

ax = axes[4]
g = sns.histplot(
    data=pd.DataFrame(pd.DataFrame(
        days[1].groupby('uid').size()).to_records()),
    ax=ax,
    x='0',
)
plt.savefig("../Thesis/figures/histplots_appid_duration_upflow_downflow_lines.pdf",
            dpi=400, bbox_inches='tight', pad_inches=0.005)
```

附录 F 问题一：量化指标代码

```
#!/usr/bin/env python
# coding: utf-8

# # 载入套件

# In[ ]:


from typing import Dict
import numpy as np
import pandas as pd

import ydata_profiling
import scienceplots
import matplotlib.pyplot as plt
import seaborn as sns
import matplotlib

get_ipython().run_line_magic('matplotlib', 'inline')

pd.plotting.register_matplotlib_converters()
plt.style.use(['fivethirtyeight', 'science', 'grid'])
sns.set(
    rc={'text.usetex': True},
    font="serif",
    font_scale=1.2
)

# In[ ]:

days = {i: pd.read_pickle(f'../Datasets/day{str(i).zfill(2)}.pkl')
        for i in range(1, 22)}
```

```
# # 使用时长

# In[ ]:

dfs = []
value = 'duration'
for i in range(1, 8):
    pt = days[i].pivot_table(
        index='uid',
        columns='app_class',
        values=value,
        aggfunc=np.sum,
        fill_value=0
    )

    feature = pd.DataFrame(pt.to_records())
    feature.columns = [
        f'{value}_{i}' if i != 'uid' else i for i in feature.columns]
    feature.drop(columns=f'{value}_NaN', inplace=True)
    feature['day'] = i
    dfs.append(feature)

feature_duration = pd.concat(dfs, ignore_index=True).groupby([
    'uid']).sum().drop(columns='day')

feature_duration

# # 上行流量

# In[ ]:

dfs = []
value = 'up_flow'
for i in range(1, 8):
```

```
pt = days[i].pivot_table(
    index='uid',
    columns='app_class',
    values=value,
    aggfunc=np.sum,
    fill_value=0
)

feature = pd.DataFrame(pt.to_records())
feature.columns = [
    f'{value}_{i}' if i != 'uid' else i for i in feature.columns]
feature.drop(columns=f'{value}_NaN', inplace=True)
feature['day'] = i
dfs.append(feature)

feature_upflow = pd.concat(dfs, ignore_index=True).groupby([
    'uid']).sum().drop(columns='day')

feature_upflow

# # 下行流量

# In[ ]:

dfs = []
value = 'down_flow'
for i in range(1, 8):
    pt = days[i].pivot_table(
        index='uid',
        columns='app_class',
        values=value,
        aggfunc=np.sum,
        fill_value=0
)

    feature = pd.DataFrame(pt.to_records())
```

```
feature.columns = [
    f'{value}_{i}' if i != 'uid' else i for i in feature.columns]
feature.drop(columns=f'{value}_NaN', inplace=True)
feature['day'] = i
dfs.append(feature)

feature_downflow = pd.concat(dfs, ignore_index=True).groupby([
    'uid']).sum().drop(columns='day')

feature_downflow

# # 使用频次

# dfs = []
# value = 'count'
# for i in range(1, 8):
#     pt = days[i].pivot_table(
#         index='uid',
#         columns='app_class',
#         values='duration',
#         aggfunc='count',
#         fill_value=0
#     )
#     #
#     feature = pd.DataFrame(pt.to_records())
#     feature.columns = [
#         f'{value}_{i}' if i != 'uid' else i for i in feature.columns]
#     feature.drop(columns=f'{value}_NaN', inplace=True)
#     feature['day'] = i
#     dfs.append(feature)
#     #
# feature_count = pd.concat(dfs, ignore_index=True).groupby([
#     'uid']).sum().drop(columns='day')

# feature_count

# # 特征融合
```

```
# 前7日各类APP使用时长、使用频次、上行流量、下行流量

# In[ ]:

features = pd.DataFrame(pd.concat([feature_duration, feature_upflow,
                                    feature_downflow, feature_count],
                                    axis=1).to_records())

features.to_pickle('../Datasets/features_q1.pkl')
```

附录 G 问题一：聚类算法代码

```
#!/usr/bin/env python
# coding: utf-8

# # 载入套件

# In[1]:


from sklearn.decomposition import PCA
from sklearn.cluster import KMeans
from sklearn import metrics
from sklearn.cluster import DBSCAN

from typing import Dict
from tqdm.auto import tqdm

import numpy as np
import random
import pandas as pd
import ydata_profiling

import scienceplots
import matplotlib.pyplot as plt
```

```
import seaborn as sns
import matplotlib
import matplotlib.cm as cm

import torch
import os

get_ipython().run_line_magic('matplotlib', 'inline')

pd.plotting.register_matplotlib_converters()
sns.set_style("whitegrid")
sns.set_palette("RdBu")
sns.set(
    rc={'text.usetex': True},
    font="serif",
    font_scale=1.2
)

SEED = 20230723

def same_seed(seed=SEED):
    random.seed(seed)
    os.environ['PYTHONHASHSEED'] = str(seed)
    np.random.seed(seed)
    torch.manual_seed(seed)
    torch.cuda.manual_seed(seed)
    torch.backends.cudnn.deterministic = True
    torch.backends.cudnn.benchmark = False
    if torch.cuda.is_available():
        torch.cuda.manual_seed_all(seed)

same_seed()

# ## 数据预处理
```

```
# In[2]:
```

```
features = pd.read_pickle('../Datasets/features_q1.pkl')
for aspect in ['duration', 'up_flow', 'down_flow', 'count']:
    cols = [i for i in features.columns if i.startswith(aspect)]
    features[cols] = (features[cols] - features[cols].values.mean()) /
        features[cols].values.std()
```

```
# ### 皮尔逊相关系数
```

```
# In[3]:
```

```
# corr = features.drop(columns='uid').corr()
# mask = np.triu(np.ones_like(corr, dtype=bool))
# f, ax = plt.subplots(figsize=(11, 9))
# sns.heatmap(corr, mask=mask, cmap=sns.color_palette("coolwarm"), vmax=.3,
#             center=0,
#             square=True, linewidths=.5, cbar_kws={"shrink": .5})

# plt.savefig("../Thesis/figures/heatmap_features.pdf",
#             dpi=400, bbox_inches='tight', pad_inches=0.005)
```

```
# ### 主成分分析
```

```
# In[4]:
```

```
# pca = PCA()
# pca.fit(features.drop(columns='uid'))
# fs = np.arange(pca.n_components_)[::40]

# plt.figure(figsize=(10, 3))
```

```
# g = sns.barplot(x=fs, y=pca.explained_variance_[:40])

# plt.savefig("../Thesis/figures/barplot_pca_features.pdf",
#             dpi=400, bbox_inches='tight', pad_inches=0.005)

# In[5] :

pca = PCA(n_components=3)
pca.fit(features.drop(columns='uid'))
transformed = pca.transform(features.drop(columns='uid'))
X = transformed

# # 聚类分析

# ## 原型聚类: K-Means++

# In[6] :

# Ks = range(2, 52)
# inertias = []
# scores = []

# for k in Ks:
#     model = KMeans(n_clusters=k, random_state=0, n_init="auto", max_iter=1000)
#     model.fit(X)
#     inertias.append(model.inertia_)
#     scores.append(metrics.calinski_harabasz_score(X, model.labels_))

# fig, axes = plt.subplots(1, 2, figsize=(12, 5.5))

# g = sns.lineplot(x=Ks, y=inertias, marker='o', ax=axes[0])
# g.set(
#     xlabel='number of clusters',
```

```
#     ylabel='inertia'
# )

# g = sns.lineplot(x=Ks, y=scores,
#                     marker='o', ax=axes[1])
# g.set(
#     xlabel='number of clusters',
#     ylabel='calinski harabasz score'
# )

# plt.savefig("../Thesis/figures/lineplot_Ks_inertias_CH.pdf",
#             dpi=400, bbox_inches='tight', pad_inches=0.005)

# In[7]:


labels_kmeans = KMeans(n_clusters=5, random_state=0,
                       n_init="auto", max_iter=1000).fit_predict(transformed)
labels_kmeans

# In[8]:


labels_kmeans.shape

# ## 密度聚类: DBSCAN

# In[15]:


# from sklearn.neighbors import NearestNeighbors

# # ref:
#     https://stackoverflow.com/questions/15050389/estimating-choosing-optimal-hyperparameters
```

```
# k = 2 * X.shape[-1] - 1
# nbrs = NearestNeighbors(n_neighbors=k, radius=1.0).fit(X)
# distances, indices = nbrs.kneighbors(X)
# distances = np.sort(distances, axis=0)
# distances = distances[:, k-1]

# g = sns.relplot(
#     distances,
#     kind='line'
# )

# plt.axhline(y=2, linestyle=':')

# g.set(
#     xlabel='Index of Users in the dataset',
#     ylabel='Sorted {}-nearest Neighbor Distance'.format(k)
# )

# plt.savefig("../Thesis/figures/relplot_knn_distances.pdf",
#             dpi=400, bbox_inches='tight', pad_inches=0.005)

# In[16]:


db = DBSCAN(eps=2, min_samples=6).fit(X)
labels_dbSCAN = db.labels_

# Number of clusters in labels, ignoring noise if present.
n_clusters_ = len(set(labels_dbSCAN)) - (1 if -1 in labels_dbSCAN else 0)
n_noise_ = list(labels_dbSCAN).count(-1)

print(f"Estimated number of clusters: {n_clusters_}")
print(f"Estimated number of noise points: {n_noise_}")

# In[17]:
```

```
set(labels_dbSCAN)

# In[18]:


counter = {}
for i in labels_dbSCAN:
    counter[i] = counter.get(i, 0)+1
counter

# ## 层次聚类: BRICH

# In[11]:


from sklearn.cluster import Birch
import numpy as np
from matplotlib import pyplot as plt
from scipy.cluster.hierarchy import dendrogram
from sklearn.cluster import AgglomerativeClustering

brc = Birch(n_clusters=None)
brc.fit(transformed)
labels_brc = brc.labels_
centroids = brc.subcluster_centers_

model = AgglomerativeClustering(n_clusters=6)
model = model.fit(centroids)

# In[12]:
```

```
# def plot_dendrogram(model, **kwargs):
#     # Create linkage matrix and then plot the dendrogram

#     # create the counts of samples under each node
#     counts = np.zeros(model.children_.shape[0])
#     n_samples = len(model.labels_)
#     for i, merge in enumerate(model.children_):
#         current_count = 0
#         for child_idx in merge:
#             if child_idx < n_samples:
#                 current_count += 1 # leaf node
#             else:
#                 current_count += counts[child_idx - n_samples]
#         counts[i] = current_count

#     linkage_matrix = np.column_stack(
#         [model.children_, model.distances_, counts]
#     ).astype(float)

#     # Plot the corresponding dendrogram
#     dendrogram(linkage_matrix, **kwargs)

# plt.figure(figsize=(15, 15))
# plt.title("Hierarchical Clustering Dendrogram")
# # plot the top p levels of the dendrogram
# plot_dendrogram(model, truncate_mode="level", p=6)
# plt.xlabel("Number of points in node")
# plt.xticks(rotation=90)
# plt.savefig("../Thesis/figures/dendrogram.pdf",
#             dpi=400, bbox_inches='tight', pad_inches=0.005)

# In[13]:


labels_brc_agn = []
```

```
for i in labels_brc:  
    labels_brc_agn.append(model.labels_[i])  
  
# In[14]:  
  
set(labels_brc_agn)  
  
# # 算法比较  
  
# ## 内部评价指标  
  
# In[17]:  
  
data = {  
    'Method': ['K-Means++', 'DBSCAN', 'BIRCH+AGNES'],  
    'Calinski-Harabasz': [],  
    'Davies-Bouldin': [],  
    'Silhouette': []  
}  
  
for labels in [labels_kmeans, labels_dbSCAN, labels_brc_agn]:  
    data['Calinski-Harabasz'].append(  
        metrics.calinski_harabasz_score(transformed, labels))  
    data['Davies-Bouldin'].append(  
        metrics.davies_bouldin_score(transformed, labels))  
    data['Silhouette'].append(  
        metrics.silhouette_score(transformed, labels))  
  
df = pd.DataFrame(data)
```

```
# In[ ]:

print(df.to_latex())


# ## 3D


# In[ ]:


fig = plt.figure(figsize=(10, 10))
ax = fig.add_subplot(projection='3d')

ys = transformed[:, 0]
xs = transformed[:, 1]
zs = transformed[:, 2]
ax.scatter(xs, ys, zs,
           color=[f'C{i+1}' for i in labels_brc_agn])

ax.set_xlabel('X')
ax.set_ylabel('Y')
ax.set_zlabel('Z')

plt.savefig("../Thesis/figures/3d_brc_agn.png",
           dpi=500, bbox_inches='tight', pad_inches=0.005)


# ## t-SNE


# In[20]:


# from sklearn.manifold import TSNE

# # 大概需要运行5分钟
# X_embedded = TSNE(n_components=2, learning_rate='auto',
```

```
#                 random_state=0).fit_transform(transformed)
# np.save('t_SNE_X.npy', X_embedded)

X_embedded = np.load('../Datasets/t_SNE_X.npy')
x, y = X_embedded[:, 0], X_embedded[:, 1]

# In[22] :

label = labels_dbSCAN
plt.figure(figsize=(6, 6))
sns.scatterplot(
    x=x,
    y=y,
    hue=label,
    size=0.2,
    style=label,
    alpha=0.75,
)
plt.legend([], [], frameon=False)
plt.savefig("../Thesis/figures/2d_dbSCAN.png",
           dpi=200, bbox_inches='tight', pad_inches=0.005)

# In[ ]:
```

附录 H 问题一：用户画像代码

```
#!/usr/bin/env python
# coding: utf-8

# # 载入套件

# In[1]:
```

```
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans
from sklearn import metrics
from sklearn.cluster import DBSCAN

from typing import Dict
from tqdm.auto import tqdm

import numpy as np
import random
import pandas as pd
import ydata_profiling

import scienceplots
import matplotlib.pyplot as plt
import seaborn as sns
import matplotlib
import matplotlib.cm as cm

import torch
import os

get_ipython().run_line_magic('matplotlib', 'inline')

pd.plotting.register_matplotlib_converters()
sns.set_style("whitegrid")
sns.set_palette("RdBu")
sns.set(
    rc={'text.usetex': True},
    font="serif",
    font_scale=1.2
)
SEED = 20230723
```

```
def same_seed(seed=SEED):
    random.seed(seed)
    os.environ['PYTHONHASHSEED'] = str(seed)
    np.random.seed(seed)
    torch.manual_seed(seed)
    torch.cuda.manual_seed(seed)
    torch.backends.cudnn.deterministic = True
    torch.backends.cudnn.benchmark = False
    if torch.cuda.is_available():
        torch.cuda.manual_seed_all(seed)

same_seed()

features = pd.read_pickle('../Datasets/features_q1.pkl')
for aspect in ['duration', 'up_flow', 'down_flow', 'count']:
    cols = [i for i in features.columns if i.startswith(aspect)]
    features[cols] = (features[cols] - features[cols].values.mean()) /
        features[cols].values.std()

pca = PCA(n_components=3)
pca.fit(features.drop(columns='uid'))
transformed = pca.transform(features.drop(columns='uid'))
X = transformed

model = KMeans(n_clusters=5, random_state=0,
                n_init="auto", max_iter=1000)
labels_kmeans = model.fit_predict(transformed)
labels_kmeans

# In[2]:
```



```
features = pd.read_pickle('../Datasets/features_q1.pkl')
features['cluster'] = labels_kmeans
features['cluster'] = features['cluster'].apply(lambda x: str(x))
```

```
features

# # 可视化

# In[3]:


fig, axes = plt.subplots(4, 5, figsize=(35, 25))
aspect = 'duration'

for ax, y in zip(axes.flatten(), [i for i in features.columns if
i.startswith(aspect)]):
    sns.stripplot(
        ax=ax,
        data=features,
        x='cluster',
        y=y,
        hue="cluster",
        size=1.2,
        alpha=0.95
    )

    ax.legend([], [], frameon=False)

plt.savefig("../Thesis/figures/striplott_duration_cluster.png",
            dpi=200, bbox_inches='tight', pad_inches=0.005)


# In[4]:


fig, axes = plt.subplots(4, 5, figsize=(35, 25))
aspect = 'count'

for ax, y in zip(axes.flatten(), [i for i in features.columns if
i.startswith(aspect)]):
```

```
sns.stripplot(  
    ax=ax,  
    data=features,  
    x='cluster',  
    y=y,  
    hue="cluster",  
    size=1.2,  
    alpha=0.95  
)  
  
ax.legend([], [], frameon=False)  
  
plt.savefig("../Thesis/figures/stripplot_count_cluster.png",  
           dpi=200, bbox_inches='tight', pad_inches=0.005)  
  
# In[5]:  
  
fig, axes = plt.subplots(4, 5, figsize=(35, 25))  
aspect = 'up_flow'  
  
for ax, y in zip(axes.flatten(), [i for i in features.columns if  
i.startswith(aspect)]):  
    sns.stripplot(  
        ax=ax,  
        data=features,  
        x='cluster',  
        y=y,  
        hue="cluster",  
        size=1.2,  
        alpha=0.95  
)  
  
    ax.legend([], [], frameon=False)  
  
plt.savefig("../Thesis/figures/stripplot_up_flow_cluster.png",
```

```
dpi=200, bbox_inches='tight', pad_inches=0.005)

# In[6]:


fig, axes = plt.subplots(4, 5, figsize=(35, 25))
aspect = 'down_flow'

for ax, y in zip(axes.flatten(), [i for i in features.columns if
i.startswith(aspect)]):
    sns.stripplot(
        ax=ax,
        data=features,
        x='cluster',
        y=y,
        hue="cluster",
        size=1.2,
        alpha=0.95
    )

    ax.legend([], [], frameon=False)

plt.savefig("../Thesis/figures/stripplot_down_flow_cluster.png",
            dpi=200, bbox_inches='tight', pad_inches=0.005)

# # 数值

# In[7]:


features[['cluster']].value_counts()

# In[8]:
```

```
features.shape

# In[9]:


for aspect in ['duration', 'count', 'up_flow', 'down_flow']:
    features[f'{aspect}_sum'] = 0
    for i in [i for i in features.columns if i.startswith(aspect)]:
        features[f'{aspect}_sum'] += features[i]

# In[10]:


pt = features.drop(columns='uid').pivot_table(
    index='cluster',
    aggfunc=np.mean
)

c = 'a'
pt[[f'count_{c}', f'duration_{c}', f'up_flow_{c}', f'down_flow_{c}']]

# In[11]:


x = range(20)
plt.figure(figsize=(10, 5))
for i in range(5):
    y = pt[[i for i in features.columns if i.startswith(
        'duration') and 'sum' not in str(i)]].iloc[i]
    plt.plot(x, y, c=f'C{i}', label=i, alpha=0.75)
plt.xticks(range(20), list("abcdefghijklmnopqrstuvwxyz"))
plt.legend()
plt.savefig("../Thesis/figures/plot_profile.pdf",
```

```
dpi=200, bbox_inches='tight', pad_inches=0.005)
```

附录 I 问题二：训练资料代码

附录 J 问题二：分类问题代码

附录 K 问题二：回归问题代码

附录 L 问题二：模型评价代码
