

2023 年第二届“钉钉杯”大学生 大数据挑战赛论文

题 目 "明镜高悬，倩影悠然": 智能手机用户行为的预测与推荐系统构建

摘 要

本研究致力于深入分析和预测智能手机用户的行为模式。在数据处理阶段，用户应用使用行为数据经过清洗和特征工程处理，同时通过肘部法则和轮廓系数法来确定最优的聚类数量。在此基础上，实施了四种不同的聚类方法，包括 K-Means、DBSCAN、Agglomerative Clustering 和 Gaussian Mixture。结果显示，K-Means 算法的表现最为优异，成功刻画了用户的行为模式。

基于 K-Means 聚类结果，本研究构建了五类用户画像，包括“分享达人”、“资讯狂热者”、“社交巨头”、“轻度用户”和“流量吞噬者”，并对每个用户群体进行了详细的特性分析。这些用户画像全面描绘了用户的属性、偏好和行为习惯，为理解用户需求和优化产品提供了有价值的洞察。

在 APP 使用情况预测部分，本研究尝试了多种机器学习和深度学习算法，包括逻辑回归、K-近邻（KNN）、朴素贝叶斯、随机森林、梯度提升树（GBDT）、XGBoost、AdaBoost、BP 神经网络、卷积神经网络（CNN）、LSTM 等。在分类问题上，AdaBoost 模型表现最为突出，达到了 0.9968 的准确率。对于预测用户应用使用时长的回归问题，BP 神经网络算法取得了最好的效果，其测试集标准化均方误差（NMSE）为 0.1725。在构建推荐系统时，我们考虑了基于内容的推荐、协同过滤算法、Singular Value Decomposition 以及 Neural Collaborative Filtering。神经协同过滤(NCF)在预测用户对 APP 的使用偏好方面取得了 0.9762 的准确率，同时还采用了 MAE、Coverage、Novelty 和 Popularity Bias 等评价指标来全面评估模型性能。

本研究深度分析了智能手机用户行为数据，揭示其使用特征，但仍存在局限。在数据处理、聚类、模型应用及推荐系统等环节都有优化潜力。未来研究可引入更先进的技术和方法，以增强分析和预测的准确性。总体而言，本研究为智能手机用户行为分析提供了方向，期待技术和研究的进一步融合，为此领域带来更深入的见解。

关键词：用户画像；深度学习；集成算法；神经协同过滤；推荐系统

目 录

摘 要	1
第一章 绪论	1
1.1 问题背景	1
1.2 文献综述	1
第二章 问题重述与分析	4
2.1 问题重述	4
2.2 问题分析	4
2.2.1 APP 使用行为洞察与预测	4
2.2.2 智能 APP 推荐策略设计	4
第三章 数据预处理与分析	5
3.1 数据集概述	5
3.2 数据预处理策略	5
3.3 特征筛选及优化	8
第四章 方法与技术	9
4.1 用户聚类分析与优化	9
4.2 智能手机 APP 使用情况预测分析	9
4.3 APP 推荐策略与评估	10
4.4 模型与算法选择理论依据	10
4.5 模型训练与验证流程	11
第五章 结果展示	13
5.1 聚类分析结果展示	13
5.2 用户画像构建与特征分析	16
5.3 分类预测结果展示	16
5.4 回归预测结果展示	20
5.4 推荐系统预测结果展示	25
第六章 讨论与深度分析	27
6.1 深度洞察与模型解读	27
6.1.1 智能手机用户行为模式解读	27
6.1.2 用户应用使用情况预测分析	27
6.2 研究限制与潜在改进	28
第七章 结论与未来研究方向	30
7.1 研究成果与贡献	30
7.2 未来研究方向展望	30
参考文献	31
附录	34

第一章 绪论

1.1 问题背景

随着信息技术的飞速进步，智能手机已经普遍渗透到人们的日常生活中。特别是在全球最大的智能手机市场——中国，各类应用程序（APP）如雨后春笋般涌现。包括但不限于游戏、购物、社交、资讯、理财等 APP，这些已成为现代生活的核心组成部分，同时也为用户行为数据提供了大量的生成源。这些海量的数据为产品优化及市场决策提供了极其重要的参考价值^[1]。例如，某些流行的社交媒体 APP，如微信和微博，就广泛利用了用户行为数据进行产品优化和个性化推荐。这些 APP 通过分析用户的行为模式（如用户通常何时在线，用户偏好的内容类型，用户与哪些其他用户互动等），可以生成个性化的内容推荐，以提高用户的使用频率和停留时间。此外，用户行为数据还可以用来发现潜在的产品问题。例如，如果数据显示用户在尝试使用某个功能后频繁退出 APP，这可能表明该功能存在用户界面问题，需要进一步优化^[2]。

在大数据技术推动下，智能手机用户监测数据分析已经成为研究热点。本研究即基于此背景，着眼于某公司连续 30 天的 4 万多智能手机用户的监测数据进行深入分析。

本研究重点关注两个问题：

APP 使用预测：利用用户 1 至 15 天的 20 类 APP 数据，构建模型预测第 16 至 30 天的 APP 类别和日均使用时长。预测的准确性通过与实际数据比对，采用准确率和 NMSE 为主要评估指标。为确保结果的可靠性，测试集不参与训练和验证。

APP 推荐系统：专注于 app_class.csv 中的常用 APP，基于用户整月数据，设计推荐系统为用户提供最佳 APP 选择，并对系统的构建与评估进行描述。

解决这两个问题将有助于更深入地理解用户的行为模式，为 APP 的设计和优化提供决策依据，同时也为市场策略制定、产品推荐、广告投放以及公共服务等领域提供有价值的参考信息。

1.2 文献综述

智能手机的广泛应用和普及对中国智能手机市场产生了显著影响，催生了手机软件的快速发展。根据中国互联网络信息中心(CNNIC)发布的数据，截至 2022 年 6 月，中国网民规模达到了 10.51 亿，互联网普及率为 74.4%^[3]。这个趋势意味着智能手机的使用已经普及到各个年龄段和社会阶层。

智能手机上众多的应用程序(APP)提供了丰富多元的服务，满足了用户的日常需求，

并形成了大量的用户行为数据。对这些数据的深度挖掘和分析，能够揭示用户的行为模式，预测用户的需求，从而对产品优化及市场决策具有重大的指导价值。

随着移动互联网技术的创新和发展，以及大数据、人工智能等新技术的广泛应用，智能手机用户监测数据分析领域得到了快速发展。研究者们已经从多个角度和层面探讨了智能手机用户监测数据分析的相关问题，包括用户分类、用户活动分类以及用户对 APP 的偏好和需求等。

在 APP 用户聚类 and 用户画像研究领域，最常用的聚类算法是 K-均值算法和谱聚类算法。然而，这些算法在处理手机 APP 用户数据时有其局限性。例如，K-means 算法虽然实现简单，但需要预先确定簇的个数，并且对初始中心点敏感。而谱聚类算法可以发现非线性可分的簇，但其计算复杂度较高，并且需要选择合适的相似度度量和降维方法。

为提高聚类效果和效率，研究者们已经对这些算法进行了改进和优化。例如，毛建军基于改进 K-Means 聚类算法对移动 5G 手机用户进行了分类^[2]；马创等将 SOM 和 Kmeans 聚类算法相结合用于移动手机用户分群^[4]；童婉迪利用有效性指标 CH 值来选取 K 值，粒子群算法选取初始中心点来优化 K-means 算法^[5]；韩迪等提出了一种增量学习的优化算法，在不重新建模的情况下对 app 使用进行预测^[6]。

此外，聚类特征是手机 APP 用户聚类的关键因素，可以反映用户的 APP 使用行为和偏好。现阶段，常用的聚类特征有基于时间、频率、时长、类型等维度的特征。这些特征可以从用户的 APP 使用记录中直接提取或计算得到。有些研究者还尝试构建更高层次或更细粒度的特征，以捕捉用户的潜在需求或行为模式。例如，蒋孜文提出了 AppUsage2Vec 模型，通过学习 App 特征的上下文关系，将 App 使用序列转化为隐向量作为特征^[7]；吴志敏等采用多维度特征方法来提高分类的准确度^[8]。

总的来说，手机 APP 用户聚类 and 用户画像是一个有价值且有挑战性的研究领域。虽然已经取得了一定的成果，但仍然存在一些问题和不足，如数据质量和规模的限制、聚类算法和指标的适应性和稳定性、聚类结果的解释性和可视化等。有必要进一步深入探索手机 APP 用户行为的本质规律和内在机制，提出更有效和智能的聚类方法和指标，并将其应用于更广泛和实际的场景中。

与此同时，精准地掌握智能手机 APP 用户的兴趣和需求，为其提供个性化的 APP 推荐，是移动互联网领域的一个重要问题。推荐算法可以根据智能手机 APP 用户的特征、行为、反馈等数据来预测手机 APP 用户对每一种 APP 的兴趣程度，并生成相应的推荐列表。

常见的推荐算法有基于内容的过滤、基于协同过滤的过滤、基于知识的过滤、基于社会化的过滤等，以及将不同算法结合起来的混合过滤。Huang Xin 等人提出了一种基于内容的推荐系统重排序方法(CBR)，在保留推荐结果准确性的同时有效提高多样性^[9]；李宏志等人融合改进了内容和协同过滤来对博客进行推荐，显著提高了推荐的结果

的准确率和召回率^[10]。

为了提高推荐的效果和用户满意度,有研究者在基本的推荐系统上进行了改进和创新。Colace Francesco 利用奇异值分解属性来计算评级预测^[11],取得了不错的进展;S. Sridhar 等人在基于内容的电影推荐模型中结合了帝王蝶优化(MBO)和深度信念网络(DBN),精确率和召回率分别达到 97.35 %和 96.60 %^[12],提高了推荐的准确性和鲁棒性。

近年来,深度学习技术也被引入到推荐系统中,利用其强大的特征提取和表示能力,提高了推荐效果和效率。冯萍等人提出一种融合图神经网络的深度学习推荐模型(GCN-MLP)^[13],实现了个性化电影推荐系统;Li Bifeng 等人结合深度学习和大数据技术,研究了一种基于大规模开放在线课程(MOOC)的个性化推荐方法^[14],表现出了良好的推荐效果。卫刚等人将 SIACNN 替换多个典型新闻推荐网络中的卷积神经网络^[15],提高了推荐效果和泛化性。

本研究将通过对智能手机用户监测数据进行深度分析,揭示用户的特征、偏好、需求和行为模式,从而实现对用户的分类、描述、预测和推荐。这将为 APP 开发者和运营者提供重要的参考信息,有助于提高 APP 的设计质量和服务水平,增强 APP 的吸引力和留存率,提升 APP 的市场竞争力和盈利能力。同时,这也可以为广告投放、产品推荐、市场调研、公共服务等领域提供有价值的信息和参考。

周铠然 · 仅供求职展示

第二章 问题重述与分析

2.1 问题重述

为研究用户行为习惯提供实践场景，本研究利用 30 天的手机用户监测数据，对用户行为模式进行深度探索，以揭示用户需求和偏好。本研究关注两个主要问题：

1.APP 使用预测:利用用户第 1~15 天的数据，预测第 16~30 天可能使用的应用类别，并验证预测准确性。以及预测第 16~30 天内，用户对各类应用的日均使用时长，以 NMSE 为评价标准。

2.推荐系统构建:仅考虑 `app_class.csv` 中的 4000 个主流应用，基于用户 30 天的使用记录，为每位用户进行应用推荐。同时详述推荐系统的结构、参数及其预测效果。

2.2 问题分析

2.2.1 APP 使用行为洞察与预测

为了深入了解智能手机用户的 APP 使用习惯，首先需要对数据进行精细的预处理。这包括诸如缺失数据填补、异常值的鉴别与处理、以及数据标准化等关键步骤。这些措施确保了数据的质量，为后续分析打下坚实基础。接下来，通过创建描述用户与 APP 交互模式的特征，能够从原始数据中提取有助于模型预测的信息。

考虑到用户与 APP 的交互数据可能存在高度维度和复杂性，使用聚类方法来探索数据中的潜在结构成为一种逻辑上的选择。例如，K-Means 和 DBSCAN 能够揭示数据的内在群聚结构。针对预测用户未来的 APP 使用趋势，经典算法如逻辑回归、多项 NB 和随机森林因其稳定性和高准确性被视为首选。同时，集成方法如 XGBoost 和深度学习结构，特别是 LSTM 和 GRU 这类能够捕捉时间依赖性的模型，也将被纳入考虑范围。

2.2.2 智能 APP 推荐策略设计

在构建推荐系统之初，数据的预处理和特征工程再次扮演了关键角色。特征工程在此环节将专注于从用户的历史数据中提取与推荐强相关的特征，从而为推荐模型提供有力支持。对于模型的选择，基于内容的推荐和协同过滤方法将作为初步策略，它们主要依赖于物品的属性和用户的历史行为。而针对数据的稀疏性，矩阵分解技术，例如 SVD 和 ALS，被认为可以为用户和物品提供更为紧凑的表示。此外，由于深度学习方法的能力在捕获数据中的复杂模式方面，神经网络推荐策略，如神经协同过滤，也应被纳入考虑。在评估推荐系统的效果时，不仅仅是准确性需要被考虑，还要关注推荐的多样性、新颖性和可能的偏见，确保推荐内容既广泛又相关。

第三章 数据预处理与分析

3.1 数据集概述

本研究的数据源于某公司提供的一份连续 30 天的智能手机用户监测数据，涵盖了超过 4 万名用户的 APP 使用情况。该数据已经过脱敏处理和数据变换，以确保用户的隐私安全。

每天的数据被保存在一个文本文件（TXT）中，每个文件含有 10 个属性列，这些属性记录了用户（以用户 ID "uid" 作为唯一标识）每天使用各款 APP（以应用 ID "appid" 作为唯一标识）的具体信息，包括起始时间、使用时长、上下行流量等。

除主要数据外，还有一个 app_class.csv 的辅助表，其中包含了两列数据。第一列为 appid，表示了 4000 多个常见 APP 的类别（app_class），这些类别用 a-t 的英文字母表示，共计 20 个常用类别，而其余的 APP 由于使用频率较低，故其类别暂未定义。数据具体列的定义如表 1 所示：

表 1：用户 APP 使用情况数据表

变量编号	变量名	释义
1	uid	用户的 id
2	appid	APP 的 id
3	app_type	APP 类型
4	start_day	使用起始天，取值 1-30
5	start_time	使用起始时间
6	end_day	使用结束天
7	end_time	使用结束时间
8	duration	使用时长（秒）
9	up_flow	上行流量
10	down_flow	下行流量

这份数据集为后续的用户行为模式研究提供了丰富的信息，包括用户选择的 APP、使用时间、频率、以及数据流量等。这些信息对于后续的用户聚类分析和 APP 使用情况预测分析具有重要的价值。

3.2 数据预处理策略

在数据分析的流程中，数据预处理阶段对于建立准确且高效的预测模型是至关重要

的环节。预处理的主要目标是将原始数据转化为更易于模型识别和处理的形式，以此优化模型的预测性能。在预处理之前，通过进行探索性数据分析来深入理解数据，包括数据分布、缺失值和异常值等，这些探索性分析的结果有助于选择最恰当的预处理策略。

对于数据类型的转换，首先考虑到的是用户标识"uid"、应用标识"appid"以及应用类型"app_type"。因为这些数据的本质属性是标识符，因此被转化为字符串类型。同时，由于起始日和结束日"start_day"与"end_day"的取值范围相对较小，所以这两项数据被转化为 8 位整数类型，以优化内存使用效率。然而，使用时长"duraction"、上行流量"up_flow"以及下行流量"down_flow"的数值量级相对较大，为了保障数值处理的精确度，这些变量被转化为 32 位整数类型。数据类型转换的具体结果如表 2 所示。

表 2：数据压缩结果

Number	Column	Dtype
0	uid	object
1	appid	object
2	app_type	object
3	start_day	int8
4	start_time	object
5	end_day	int8
6	end_time	object
7	duration	int32
8	up_flow	int32
9	down_flow	int32
10	app_class	object

dtypes: int32(3), int8(2), object(6); Memory usage: 9.1+ GB

接下来进行的是数据清洗。在这个阶段，移除了使用时长、上行流量和下行流量为 0 的数据，以免这些潜在的异常值对后续分析造成影响。同时，也剔除了使用时长过短（小于 5 秒）的数据，这部分数据可能由于用户误操作或者非常规使用行为产生，不适宜纳入分析范围。经清洗后数据状态展示在图 1 中。

	uid	appid	app_type	start_day	start_time	end_day	end_time	duration	up_flow	down_flow	app_class
2	BB1657CFD3B4B7208674BEB7D26014B2	4803	usr	1	13:40:45	1	13:44:41	-0.000979	-0.109503	-0.048295	a
3	BB1657CFD3B4B7208674BEB7D26014B2	4803	usr	1	13:47:10	1	13:47:22	-0.001199	-0.109648	-0.048765	a
4	BB1657CFD3B4B7208674BEB7D26014B2	4803	usr	1	13:47:23	1	13:50:10	-0.001046	-0.106541	-0.045870	a
6	BB1657CFD3B4B7208674BEB7D26014B2	18386	usr	1	14:19:47	1	14:20:17	-0.001182	-0.090435	-0.045800	g
7	BB1657CFD3B4B7208674BEB7D26014B2	4803	usr	1	14:20:20	1	14:24:55	-0.000939	-0.081005	-0.042177	a

图 1：清洗后数据展示

为了使数据的表示更加直观且易于理解，将使用时长的单位由秒转化为分钟，同时

将流量的单位从字节转化为 MB。对于那些所属类别未知的应用，将其类别标记为 "unknown"，以便对这部分数据进行后续的处理和分析。

最后，对于 "duration"（使用时长）、"up_flow"（上行流量）和 "down_flow"（下行流量）这三个特征，进行了数据标准化处理。数据标准化是一种将原始数据转化为均值为 0、标准差为 1 的正态分布的过程，这种处理方式能有效地消除不同特征之间的量纲影响，使得不同规模和单位的数据可以在同一水平上进行比较和分析。标准化处理后数据分别如图 2、3 所示。

	uid	appid	app_type	start_day	start_time	end_day	end_time	duration	up_flow	down_flow
0	BB1657CFD3B4B7208674BEB7D26014B2	20858	usr	2	07:05:38	2	07:11:27	348	1915724	105292
1	BB1657CFD3B4B7208674BEB7D26014B2	15841	usr	2	07:12:18	2	07:12:20	2	0	0
2	BB1657CFD3B4B7208674BEB7D26014B2	15841	usr	2	07:12:22	2	07:12:22	3	0	0
3	BB1657CFD3B4B7208674BEB7D26014B2	20858	usr	2	07:12:23	2	07:14:11	108	61059	11193
4	BB1657CFD3B4B7208674BEB7D26014B2	3892	sys	2	18:10:56	2	18:11:07	11	9567	276
...	...	...	...	...	...	...	...	...	...	...
9995	EF30B47877AC911ED4820EC9AD81D10F	18000	usr	2	00:25:59	2	00:26:59	60	0	0
9996	EF30B47877AC911ED4820EC9AD81D10F	3309	usr	2	00:32:08	2	00:32:45	37	0	0

图 2：标准化后数据展示

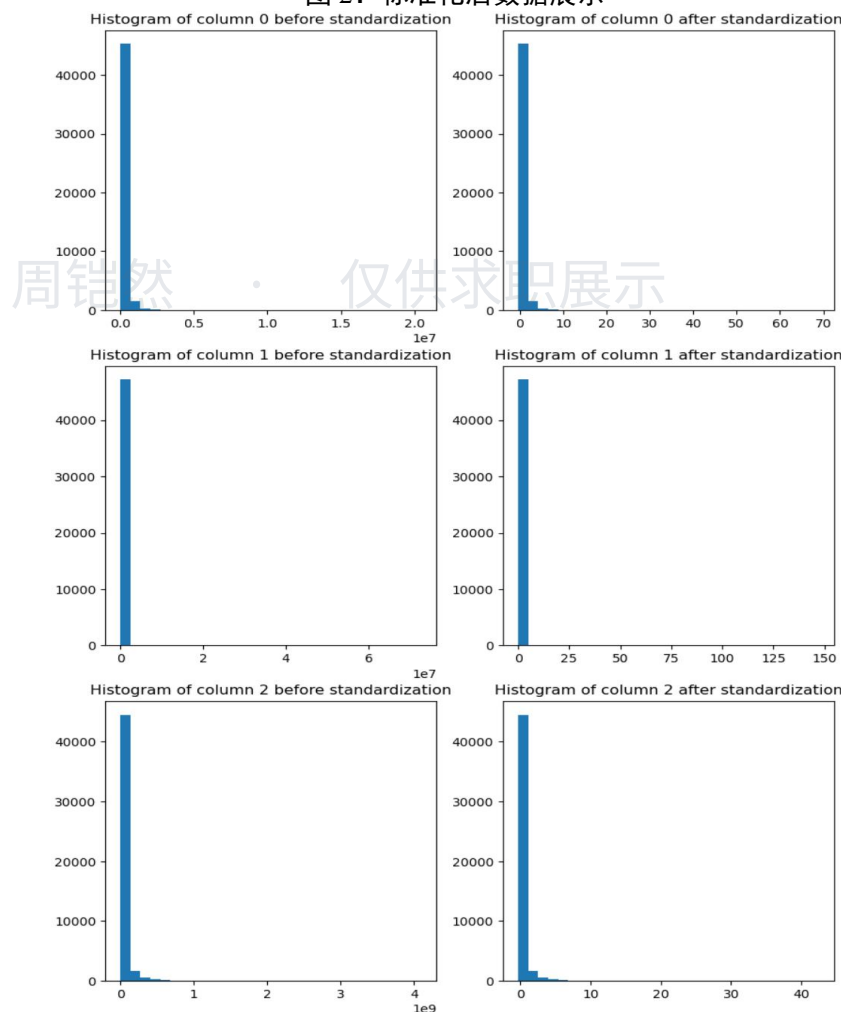


图 3：特征 "duration"、"up_flow" 和 "down_flow" 标准化前后对比

总之，经过数据预处理阶段，原始数据得以有效地转化为适合进行深度分析和模型

建立的数据集，这为后续的研究工作奠定了坚实的基础。

3.3 特征筛选及优化

特征选择与优化在预测模型的构建过程中起到关键作用，决定模型性能与预测精度。在本研究中，进行了多个特征工程步骤以优化特征集。首先，数据按用户标识("uid")和应用标识("appid")分组，并从每个组中提取了统计特征，如总使用时长、平均使用时长、总上行流量、平均上行流量、总下行流量和平均下行流量。这些特征提供了对每个用户对每个应用的使用情况的深入理解。

其次，对"uid"和"appid"进行分组，分别计算了每个用户的总使用应用数量、总使用时长、平均使用时长、总上行流量、平均上行流量、总下行流量和平均下行流量，以及每个应用的使用用户数量、总使用时长、平均使用时长、总上行流量、平均上行流量、总下行流量和平均下行流量。这些特征分别揭示了每个用户的整体 APP 使用情况和每个应用的使用频率及被使用程度。

此外，通过计算了每个用户的使用 APP 的总天数和每个 APP 被使用的总天数，可以得出用户的活跃度和 APP 的流行程度。同时，用户使用 APP 的时间跨度，即"date_span"，也被计算出来，这个新特征捕获了用户对每个 APP 的活跃期。

在特征工程的最后步骤中，将用户的总使用时长"total_duration"进行了分箱处理，将连续的使用时长变量转变为分类变量，以揭示用户使用时长的分布情况。这些工作从原始数据中提取出一系列有意义的特征，这些特征不仅能增强模型的预测能力，也为后续的数据分析和模型建立提供了强大的支持。

第四章 方法与技术

4.1 用户聚类分析与优化

首先，为确定最适合的聚类数量，本研究运用了“肘部法则”和“轮廓系数法”。这两种方法均是常见的确定最优簇数量的方法，其中“肘部法则”主要是通过找到 SSE（每个点到其所在簇中心的距离平方和）的变化率骤降的位置，而“轮廓系数法”则是通过计算所有样本的平均轮廓系数来确定。

在进行用户聚类时，主要采用了 K-Means、DBSCAN、Agglomerative Clustering 和 Gaussian Mixture 等四种聚类方法，通过这些方法将用户根据其常用 APP 的使用数据进行分群。

接下来，利用 t-SNE 降维技术将多维的聚类结果降至二维进行可视化，这使得研究者能够更直观地查看并理解聚类结果。为了评估各种聚类方法的效果，本研究计算了 silhouette_score、calinski_harabasz_score 和 davies_bouldin_score 等评估指标，并通过这些指标的结果选出了最优的聚类方法。

最后，利用最优的聚类方法对用户进行了聚类，并对聚类结果进行了深入的分析。这些分析包括计算每个簇的平均特征值等，这些结果将有助于研究者更深入地理解每个簇的用户特征。

4.2 智能手机 APP 使用情况预测分析

为精准预测用户未来的 APP 使用情况，本研究基于前期数据预处理和特征工程，深入分析并提取了与用户 APP 使用行为相关的特征。

在多分类任务中，预测用户未来使用的 APP 类型，本研究采用了逻辑回归、多项 NB 等传统机器学习算法作为预测基础。同时，考虑到数据的非线性和高维特点，集成学习方法如随机森林、GradientBoosting、AdaBoost 和 XGBoost 也进入考量范围。深度学习方面，BP 神经网络、CNN、LSTM 和 GRU 等算法为捕获数据中的复杂关系提供了支持。在模型训练和验证环节，利用交叉验证技术确保模型的泛化表现，通过比较各个模型在验证集上的准确率，选出最佳预测模型。

对于回归任务，即预测用户对各类 APP 的日均使用时长，本研究首先尝试了多元回归、岭回归和 LASSO 等基础回归方法。考虑到潜在的非线性关系，进一步尝试了决策树、随机森林及基于梯度提升的算法。在深度学习部分，BP 神经网络回归、CNN、LSTM 和 GRU 等方法同样进入对比列表。为量化各模型的预测效果，选择了 NMSE 作

为主要的评价指标，从而确定最优回归模型。

综上所述，本研究在智能手机 APP 使用情况预测上，融合了多种模型和算法，旨在确保预测的高准确性和鲁棒性。整个预测流程均遵循数据驱动原则，确保输出结果真实反映了用户的行为特点。

4.3 APP 推荐策略与评估

为在众多的 APP 选择面前，为用户推荐真正需要的 APP 是本研究追求的目标。为此，本研究结合了多种推荐算法，根据不同的场景和需求为用户提供相关的 APP 推荐。

首先，基于内容的推荐算法作为一种简单而有效的方法，此算法根据 APP 的特性和用户的偏好进行推荐。其核心在于计算 APP 之间的相似度，以此确保所推荐的内容与用户之前的偏好高度一致。紧接着，考虑到协同过滤的经典和有效性，本研究也采用了该策略。此方法依赖于用户的互动行为，通过分析用户或 APP 之间的相似性来生成相应的推荐。为了进一步提升推荐的准确性，矩阵分解技术如奇异值分解(SVD)和交替最小二乘法(ALS)被整合到推荐流程中。这些技术专门处理数据稀疏性问题，它们通过挖掘用户与 APP 交互的潜在特征，从而提取出关键的推荐信息。考虑到深度学习近年来在多个领域取得的突破，Neural Collaborative Filtering 也被纳入考虑范围，该方法融合了神经网络与传统的协同过滤，进一步加深了对用户和 APP 关系的解读。在评估推荐系统的性能时，选择了准确率、MAE、Coverage、Novelty 和 Popularity Bias 作为核心指标。这些指标不仅能够直观地反映推荐的准确性，还能确保推荐的广泛性、新颖性和多样性。

总的来说，本研究综合考虑了多种推荐策略，旨在为用户提供高质量且多样化的 APP 推荐。

4.4 模型与算法选择理论依据

在本研究中，所选取的模型和算法均基于其各自的优势，同时考虑到了数据的特性和预测任务的需求。

在聚类任务中，选择了 K-Means、DBSCAN、Agglomerative Clustering 以及 Gaussian Mixture 四种聚类方法。K-Means 算法是一种经典且高效的聚类方法，通过迭代优化实现簇内样本的高度相似性和簇间样本的明显差异性。但 K-Means 对簇形状有一定的假设，对于非球形簇或噪声敏感的数据，DBSCAN 能够更好地处理。Agglomerative Clustering 作为层次聚类方法，能够生成一系列嵌套的簇，提供更丰富的数据解释。而 Gaussian Mixture 模型可以估计每个数据点的簇属概率，为后续分析提供更多信息。

确定聚类数量的策略主要依赖于"肘部法则"和"轮廓系数法"。这两种方法都是经典的聚类评价指标，能够有效地平衡聚类的紧密性和分散性。"肘部法则"主要根据 SSE（Sum of Squared Errors）随簇数量变化的趋势来确定最佳的簇数量，而"轮廓系数法"则通过评估样本的平均轮廓系数来选择最优的簇数量。

在 APP 使用情况的预测任务中，选择了一系列机器学习、集成学习以及深度学习方法。其中，传统的机器学习模型如逻辑回归、KNN、朴素贝叶斯等在处理线性可分或近似线性可分的问题上具有优良的性能。集成学习方法如随机森林、梯度提升树、XGBoost、AdaBoost 等通过结合多个弱学习器的预测结果，可以提高模型的泛化能力，降低过拟合的风险。对于复杂的非线性问题，深度学习方法如 BP 神经网络、LSTM 等能够提供更为强大的模型表达能力。需要注意的是，虽然 SVM 算法在二分类问题中表现优秀，但是在大数据条件下，其训练时间过长且预测效果不佳，因此在本研究中并未被使用。

推荐系统的核心目标是为用户提供他们可能感兴趣的项目。基于内容的推荐方法强调了 APP 的特性和用户偏好的匹配，其核心思想是确保推荐内容与用户的历史行为高度一致。协同过滤，主要捕获用户与 APP 之间的相似性，依赖于用户的互动行为来生成推荐。而矩阵分解方法，如 SVD 和 ALS，可以有效地处理大规模稀疏数据，提取潜在特征，从而提高推荐的准确性。Neural Collaborative Filtering 等深度学习方法则融合了神经网络与传统的推荐技术，为用户提供更为精准的推荐内容。

为了全面评估推荐系统的效果，除了准确率之外，还引入了其他几种评估指标。MAE（Mean Absolute Error）能够评估预测评分与真实评分之间的误差，对于推荐系统的效果有直接的衡量意义。Coverage 则关注于推荐系统能够覆盖的项目范围，确保推荐的多样性。Novelty 强调为用户推荐新颖的内容，从而增加用户的探索性行为。Popularity Bias 则用于评估推荐系统是否过于依赖于热门项目，从而忽略了长尾的内容。综合使用这些指标，可以确保推荐的全面性、新颖性以及多样性。

总的来说，模型和算法的选择意在从不同的角度和层次对用户的 APP 使用行为进行深入理解和精准预测，以期在实际应用中达到最优的效果。

4.5 模型训练与验证流程

在本研究中，模型训练和验证的过程是一项综合任务，涵盖了各种不同的算法，包括聚类算法、经典机器算法、集成学习以及深度学习模型。

对于聚类算法，包括了 K-Means、DBSCAN、Agglomerative Clustering 和 Gaussian Mixture 四种不同的方法。在确定特征之后，利用 sklearn 库中相应的函数对每种聚类方法进行训练，然后对生成的聚类标签进行评估。评估指标包括 Silhouette Score、Calinski-Harabasz Score 和 Davies-Bouldin Score，这些指标均能从不同的角度反映出聚类

结果的质量。通过比较这些评估指标的大小，可以选择出性能最优的聚类方法。

在分类和回归部分，采用了包括 ElasticNet 和 XGBoost 在内的多种算法。这些模型的训练过程主要涉及到超参数的选择以及模型性能的评估。首先，定义模型和超参数网格，然后利用 GridSearchCV 进行网格搜索，通过交叉验证找出在训练集上表现最好的参数组合。在获取最优参数后，使用这些参数对模型进行训练，并绘制学习曲线，以便观察模型在训练集和验证集上的性能。最后，使用模型对测试集进行预测，并计算标准化均方根误差（NMSE）以评估模型的预测性能。为了更直观地展示不同参数组合对模型性能的影响，还会绘制网格搜索结果的可视化图。

对于深度学习模型，以 LSTM 为例，首先定义模型的网络结构、损失函数以及优化器。然后在设定的训练周期内，进行模型的反向传播和参数更新。在每个周期中，模型首先在训练集上进行训练，然后在验证集上进行评估。如果在连续的训练周期中，验证集上的损失没有显著下降，则触发早停机制，停止模型训练。模型训练过程中，会记录每个周期的训练损失和验证损失，以便在训练结束后绘制训练损失曲线。训练完成后，加载在验证集上表现最好的模型进行测试集预测，并计算相应的性能指标。

在模型训练和验证的过程中，可能会遇到过拟合或欠拟合等问题。为了解决过拟合问题，可以采取增加数据量、减少模型复杂度、增加正则化等策略。相反，对于欠拟合问题，可以尝试增加模型复杂度、增加特征数量或减少正则化等方法。

推荐系统在现代信息技术中起着至关重要的作用，通过分析用户的历史行为和偏好来为用户提供个性化的产品和服务推荐。在本研究采用了多种推荐算法并对其进行了详细的实验验证。首先，为了构建推荐模型，数据经过预处理，包括选择特定的用户、筛选出用户数超过 10 的 APP、删除异常值、计算评分等步骤。接着，基于用户的历史互动记录，使用 TF-IDF 技术构建了用户向量，并使用余弦相似度计算用户与每个 APP 之间的相似性。基于这种相似性，为每个用户推荐最相似的 APP，并对推荐结果进行评估。

其次，引入了基于用户的协同过滤方法，该方法通过计算用户之间的相似度，然后使用相似用户的评分来预测当前用户对未评分物品的评分，从而为用户提供推荐。此外，还引入了基于矩阵分解的 SVD 方法，该方法对用户 APP 评分矩阵进行分解，然后基于分解后的矩阵预测用户的评分，并对模型的性能进行评估。

最后，采用了神经协同过滤模型。NCF 结合了传统的协同过滤和神经网络，能够更好地捕捉用户和物品之间的非线性关系。在训练 NCF 模型时，首先定义了数据集，然后设计了模型结构，包括用户和物品的嵌入层，以及多个全连接层。模型使用均方误差作为损失函数，通过梯度下降法进行优化。在测试阶段，根据预测的评分为用户推荐物品，并计算了推荐结果的多种评价指标，如准确率、覆盖率、新颖性和流行度偏见。

总之，模型训练和验证是一个综合过程，涉及到模型选择、超参数调整以及性能评估等多个步骤，只有将这些步骤都仔细进行，才能确保模型在实际问题中达到最佳效果。

第五章 结果展示

5.1 聚类分析结果展示

针对智能手机用户的应用使用行为，本研究对数据进行了全面处理和清洗，填充缺失值，筛选特征，并计算每个用户对每种应用类别的平均使用时长、上行流量和下行流量。数据预处理完成后，为精确刻画用户行为，对这些特征数据进行标准化处理。

为选定最优聚类数量，本研究采用了肘部法则和轮廓系数法。通过分析手肘图（图 4）和轮廓系数图（图 5），视觉上确定了最优聚类数量（K=5），肘部法则和轮廓系数法可视化结果如图 4、图 5 所示。

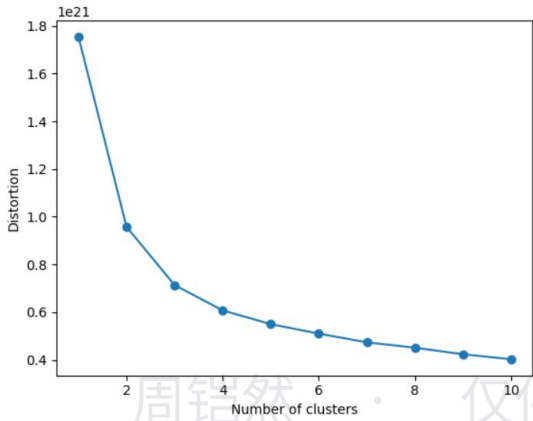


图 4：肘部法则图

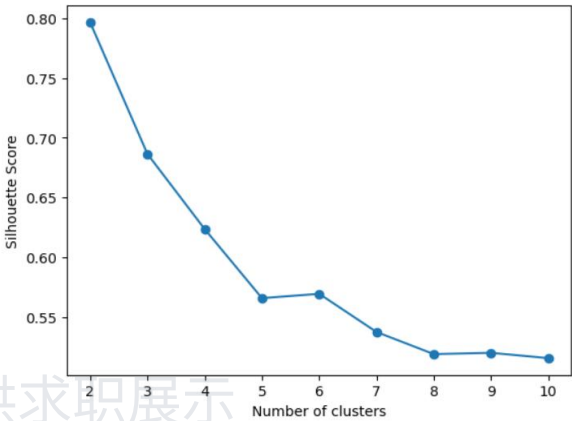


图 5：轮廓系数图

接下来，实验采用了四种不同的聚类方法，包括 K-Means，DBSCAN，Agglomerative Clustering 和 Gaussian Mixture。为了直观展现各个聚类方法的结果，通过 t-SNE 降维技术将高维数据降至二维进行可视化，聚类结果如图 6 至图 9 所示。

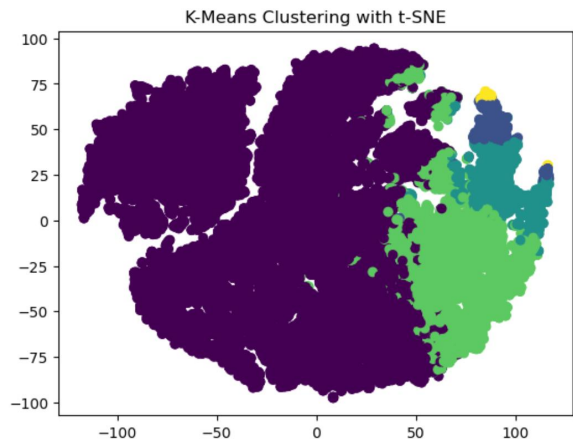


图 6：K-Means 聚类结果

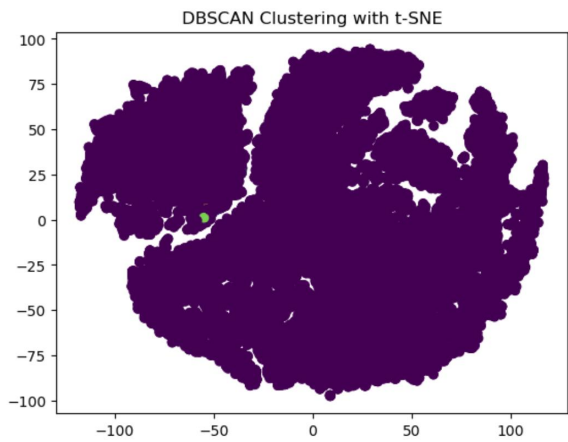


图 7：DBSCAN 聚类结果

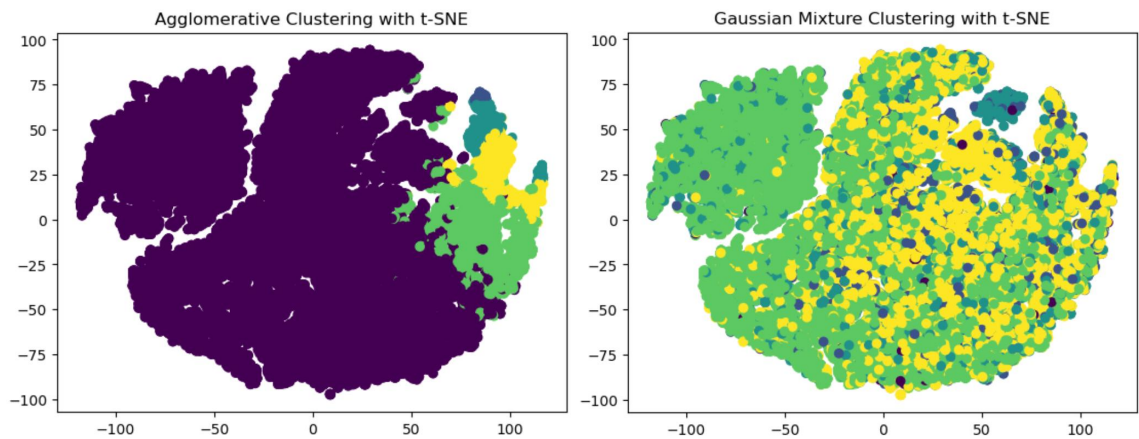


图 8: Agglomerative Clustering 聚类结果

图 9: Gaussian Mixture 聚类结果

为了直观展现各算法聚类效果，本文采用 t-SNE 降维技术将高维数据降至三维进行可视化，具体如图 10 所示：

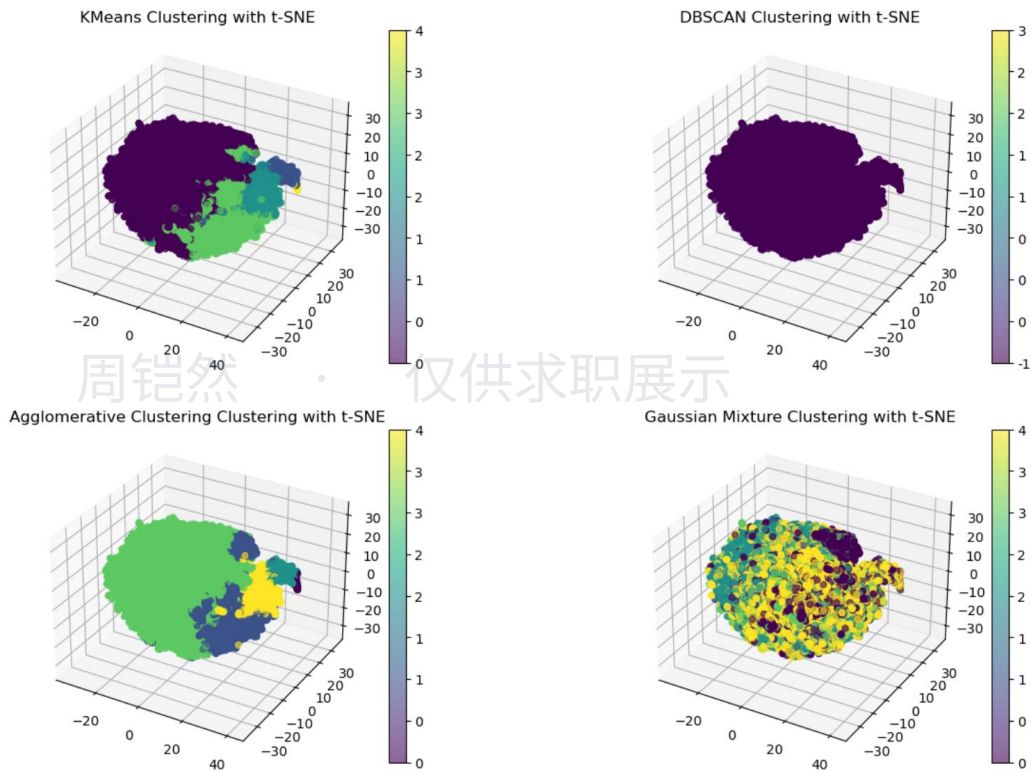


图 10: 各种聚类算法聚类效果 3D 对比图

聚类完成后，各聚类方法均经过了全面评估和比较。在此过程中，计算了各聚类方法的轮廓系数、Calinski-Harabasz 指数和 Davies-Bouldin 指数。这些度量标准从不同角度反映了聚类结果的质量，为选择最优的聚类方法提供了依据。

轮廓系数衡量了样本点与同簇其他样本点的相似性以及与其他簇的样本点的相异性。理想的聚类结果应具有较高的轮廓系数。Calinski-Harabasz 指数是类间距离与类内距离的比值，该指数越大，表示类间差异大，类内差异小，聚类效果越好。Davies-Bouldin 指数度量了每个类别的离散程度和类别间离散程度，较小的 Davies-Bouldin 指数表明聚

类效果较好。

为清晰对比不同聚类方法的效果，通过条形图表示各聚类方法的轮廓系数、Calinski-Harabasz 指数和 Davies-Bouldin 指数，见图 11 至图 13。对比结果显示，K-Means 算法在本次聚类任务中表现最优，其轮廓系数和 Calinski-Harabasz 指数均较高，而 Davies-Bouldin 指数则较低。这说明 K-Means 算法在刻画用户行为这一任务上，能够产生理想的聚类效果。

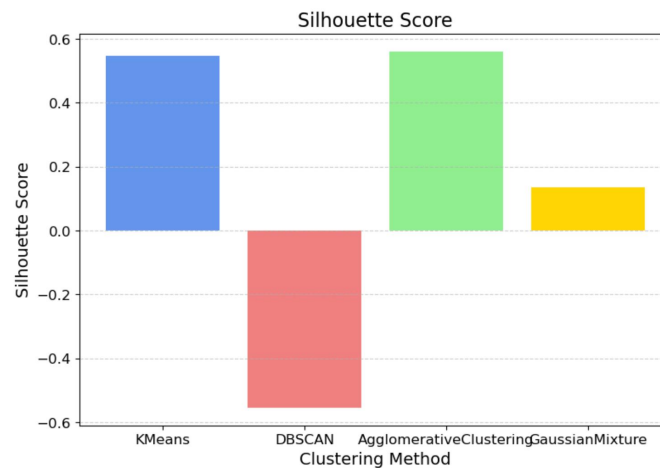


图 11：各模型轮廓系数对比

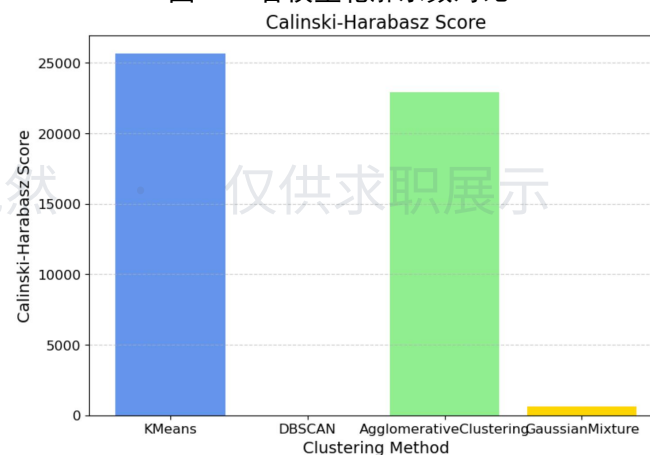


图 12：各模型 Calinski-Harabasz 指数对比

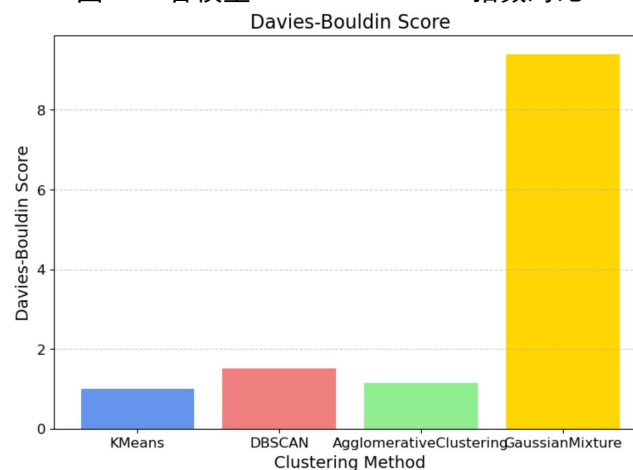


图 13：各模型 Davies-Bouldin 指数对比

5.2 用户画像构建与特征分析

基于聚类结果，构建了对应每个用户群体的用户画像，这包括了对平均使用时长、平均上行流量和平均下行流量的描述性统计。表 3 详细描绘了各用户群体的特征。

表 3：用户群体详情表

Cluster	用户标签	平均持续时间	平均上行流量	平均下行流量
1	分享达人	158.95	3055428.54	93446.11
2	资讯狂热者	107.70	9908374.26	241022.91
3	社交巨头	255.28	7121400.35	206209.54
4	轻度用户	103.10	8897740.44	238579.64
5	流量吞噬者	120.03	13242239.65	345961.38

以下是对每个用户群体的更深入的行为分析：

Cluster1: "分享达人": 这类用户的平均使用时间最短，但上行流量相对较高。这可能是由于这些用户更多地在上传内容而不是下载或观看，这表明他们可能倾向于分享而非消费内容。这种行为模式可能影响他们对社交媒体应用的选择，例如微博或知乎，他们可以在这些应用中分享自己的观点和想法。

Cluster2: "资讯狂热者": 这类用户使用时间较短，但上行和下行流量都较高。这可能意味着他们在阅读或观看大量的新闻、文章或视频，但并不经常发表评论或上传内容。因此，他们可能更喜欢使用新闻应用如今日头条或视频应用如优酷、爱奇艺。

Cluster3: "社交巨头": 这类用户具有最长的使用时间和较高的上行流量，这可能意味着他们在社交媒体上非常活跃，经常发表动态、评论或者聊天。因此，微信或 QQ 这类社交应用可能对他们更有吸引力。

Cluster4: "轻度用户": 这类用户在使用时间和流量上都较低，可能他们的手机使用主要是用于基本的通信和工作，比如打电话、发邮件等。这种行为模式可能使他们更倾向于使用高效、简洁的通讯或办公类应用，例如邮箱应用、微软 Office 等。

Cluster5: "流量吞噬者": 这类用户拥有最高的上行和下行流量，但使用时间却并不长，这可能说明他们在使用一些流量密集的应用，比如视频流应用或者大型游戏。因此，他们可能更倾向于使用游戏应用、视频播放应用等。

以上深入的用户画像分析可以帮助更好地理解用户行为，并为未来的预测和推荐提供更多的见解。

5.3 分类预测结果展示

在解决分类问题的过程中，原始数据首先被进行了精细的预处理和特征工程。例如，用户 ID("uid")经过了标签编码处理，以便将其转换为数值型数据，使得模型能够识别。

同时，所有的特征都经过了标准化处理，以确保各个特征在相同的尺度上，防止尺度问题引起模型的偏差。

在预处理步骤完成之后，进行了一系列的模型训练，包括机器学习、集成学习以及深度学习的方法，如逻辑回归、K-近邻（KNN）、朴素贝叶斯、随机森林、梯度提升树（GBDT）、XGBoost、AdaBoost、BP 神经网络以及卷积神经网络（CNN）等。为了直观地评估和比较各模型的性能，学习曲线被用来可视化模型的训练过程，并通过交叉验证以及准确率等评估指标，最终选取了最佳模型。在所有的模型中，LSTM 模型以 0.992 的准确率表现最优。

后续的图表将详细展示各个模型的训练过程和结果，以便对各模型的性能进行深入的理解和比较，各模型分类效果对比见表 4，部分模型训练过程具体见图 14 至图 27。

表 4：各模型分类效果对比表

模型	验证集准确率	验证集 Hamming 损失	测试集准确率	测试集 Hamming 损失	最佳参数
逻辑回归	0.5089	0.0326	0.5184	0.0327	{'estimator__C': 1, 'estimator__penalty': 'l2'}
多项式 NB	0.0031	0.2617	0.0052	0.2648	{'estimator__alpha': 0.01}
随机森林	0.9908	0.0004	0.9106	0.0043	{'estimator__max_depth': 10, 'estimator__n_estimators': 100}
GradientBoosting	0.9943	0.0003	0.9943	0.0003	{'estimator__learning_rate': 0.1, 'estimator__n_estimators': 100}
AdaBoost	0.9980	0.0001	0.9968	0.0002	{'estimator__learning_rate': 0.5, 'estimator__n_estimators': 50}
XGBoost	0.9980	0.0001	0.9968	0.0002	{'estimator__learning_rate': 0.5, 'estimator__n_estimators': 50}
BP 神经网络	0.0092	-	0.5620	0.0228	{'hidden_dim': 32, 'lr': 0.01}
CNN	-	-	0.7280	-	-
LSTM	-	-	0.7560	-	-
GRU	-	-	0.8420	-	-

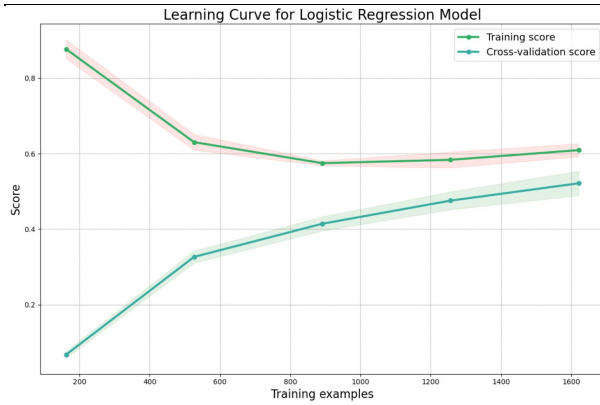


图 14: 逻辑回归训练过程



图 15: K 近邻 (KNN) 训练过程

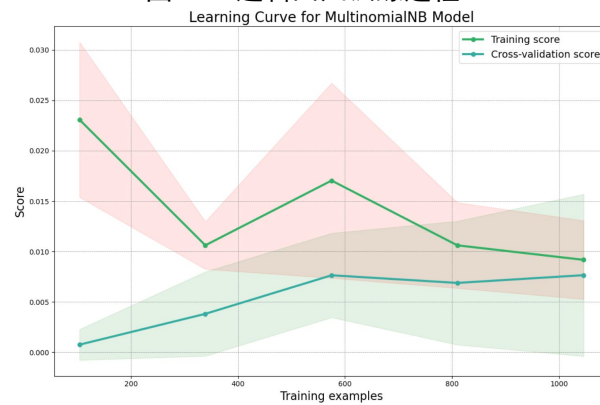


图 16: 多项式贝叶斯训练过程

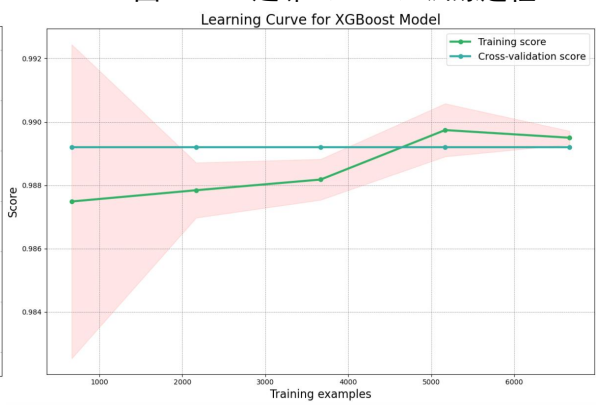


图 17: XGBoost 训练过程

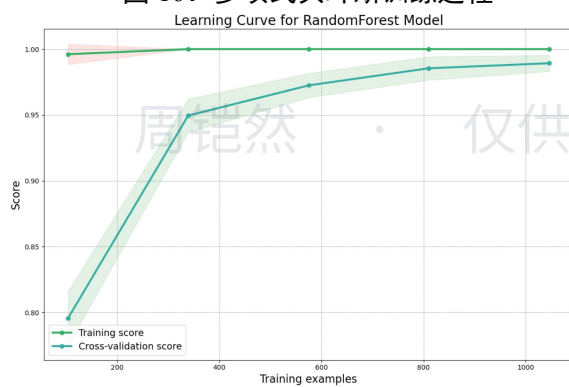


图 18: 随机森林训练过程

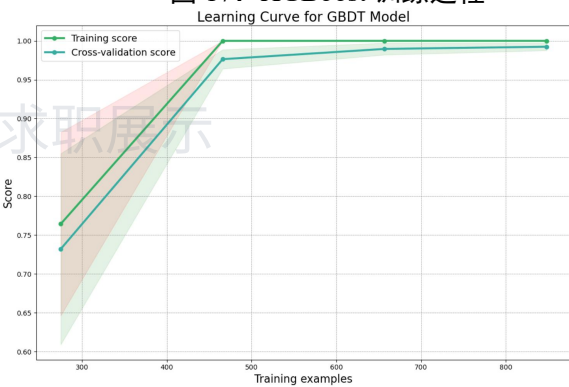


图 19: GBDT 训练过程

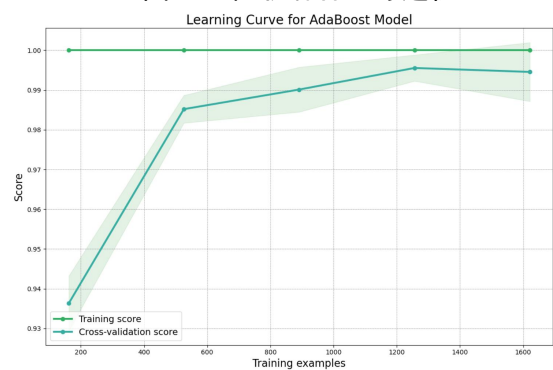


图 20: Adaboost 训练过程

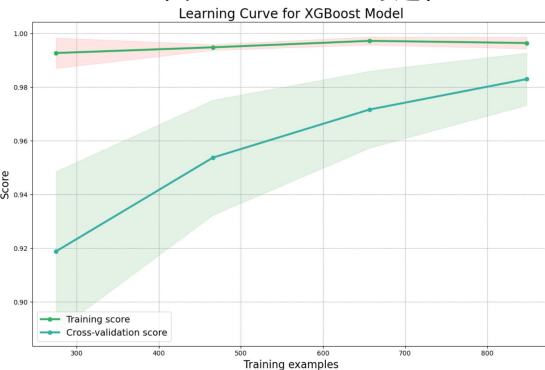


图 21: XGBoost 训练过程

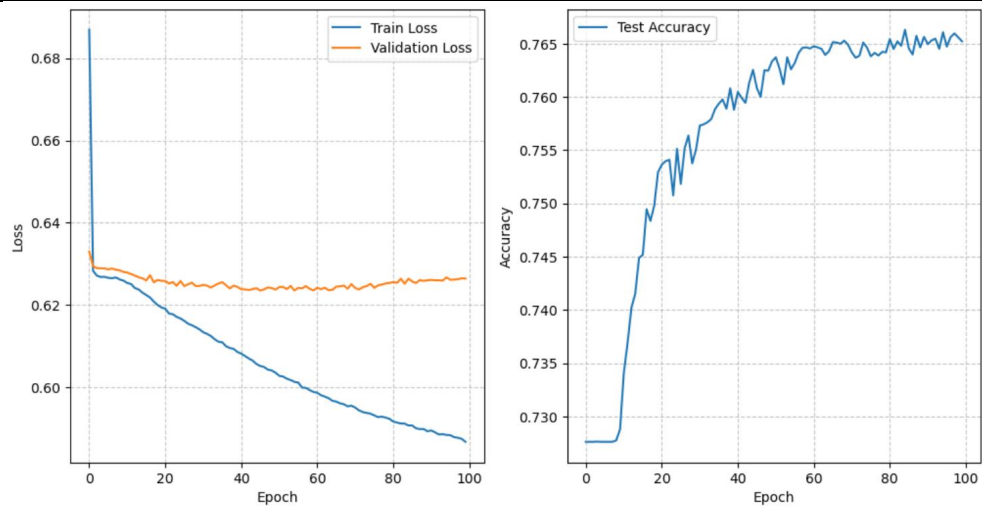


图 22: LSTM 训练过程

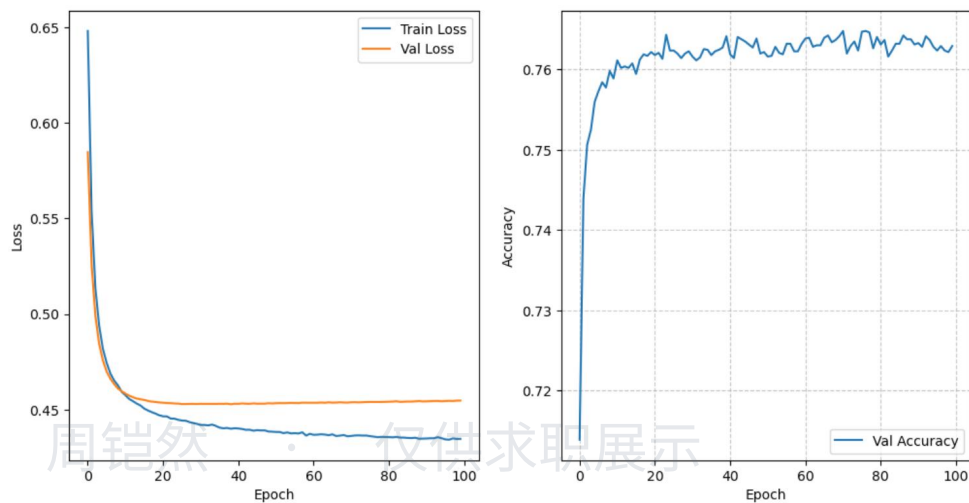


图 23: GRU 训练过程

部分模型超参数搜索过程具体见图 24 至图 27。

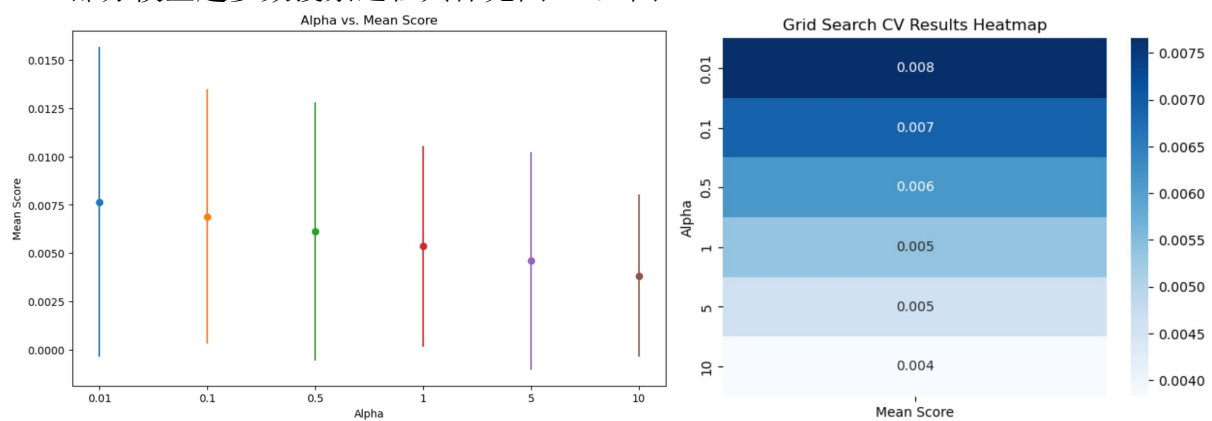


图 24: 多项 NB 超参数搜索过程

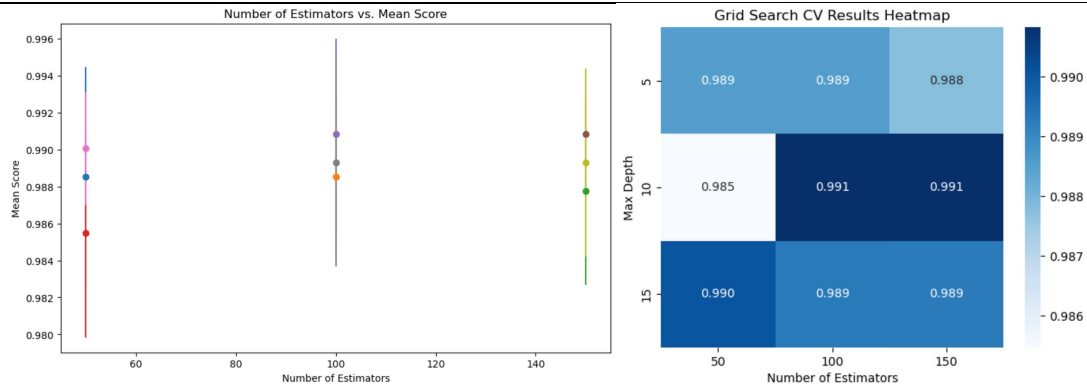


图 25: 随机森林超参数搜索过程

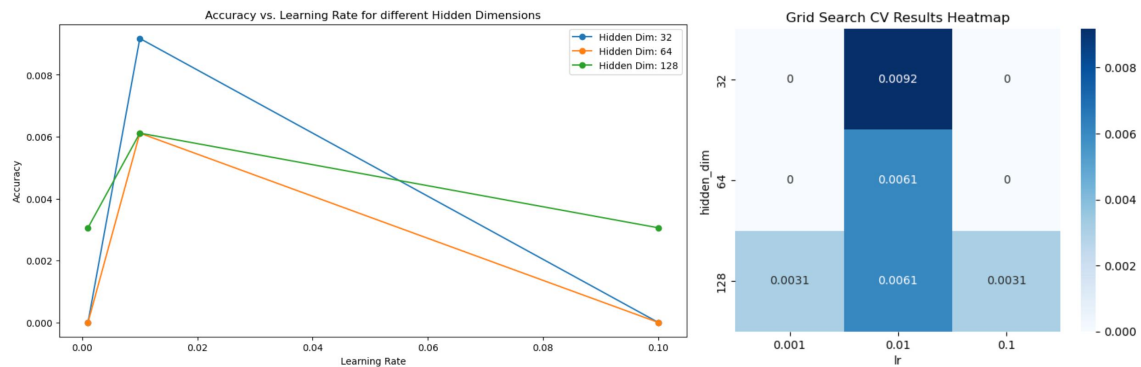


图 26: BP 神经网络超参数搜索过程

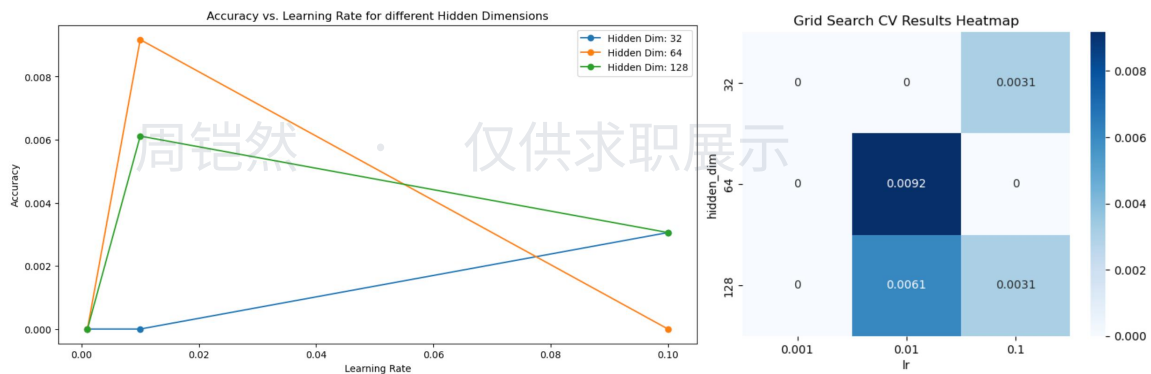


图 27: Transformer 超参数搜索过程

5.4 回归预测结果展示

针对回归问题，本研究先行进行了数据预处理和特征工程。之后，涵盖多元线性回归、岭回归、LASSO 回归、弹性网络回归（ElasticNet）、K-近邻回归（KNN）、随机森林回归、梯度提升树回归（GBDT）、XGBoost 回归、AdaBoost 回归、BP 神经网络回归，以及长短期记忆神经网络（LSTM）和门控循环单元（GRU）等多种回归模型被纳入训练范围。

在模型训练过程中，学习曲线被用于跟踪模型的学习进度。在最优模型的挑选阶段，利用交叉验证的方法，并将标准化均方误差（NMSE）设为评估指标。所有模型中，以 NMSE 评估指标表现最优的模型被选为最优模型。经过评估对比，测试集上 NMSE 最

小的模型为 BP 神经网络，其 NMSE 为 0.1725。

在接下来的图表中，将详细展示各个模型的训练过程和最终结果，包括模型在训练集和验证集上的学习曲线，以及模型在预测任务上的 NMSE 评估结果，这将为理解和比较各个模型的性能提供全面而深入的视角。图 28 至图 54 将呈现模型的部分模型训练过程和模型超参数搜索过程。

表 5：各模型回归效果对比表

模型	最佳参数	NMSE
多元回归	-	0.999355
岭回归	{'alpha': 100.0}	0.999374
LASSO	{'alpha': 0.1}	0.999359
ElasticNet 回归	{'alpha': 0.1, 'l1_ratio': 0.5}	0.999454
KNN	{'n_neighbors': 21}	0.999999
随机森林	{'max_depth': 10, 'max_features': 'sqrt', 'min_samples_leaf': 4, 'min_samples_split': 5, 'n_estimators': 500}	0.998454
Adaboost	{'learning_rate': 0.01, 'n_estimators': 50}	1.000002
XGBoost	{'learning_rate': 0.01, 'max_depth': 3, 'n_estimators': 200}	1.000003
GradientBoosting	{'learning_rate': 0.01, 'max_depth': 3, 'n_estimators': 200}	0.975918
BP 神经网络回归	-	0.172500
LSTM	-	0.798654
GRU	-	1.000000
Transformer	-	1.000000

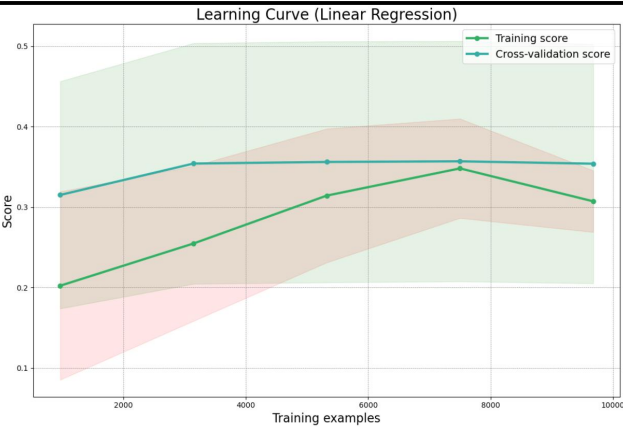


图 28：多元回归训练过程

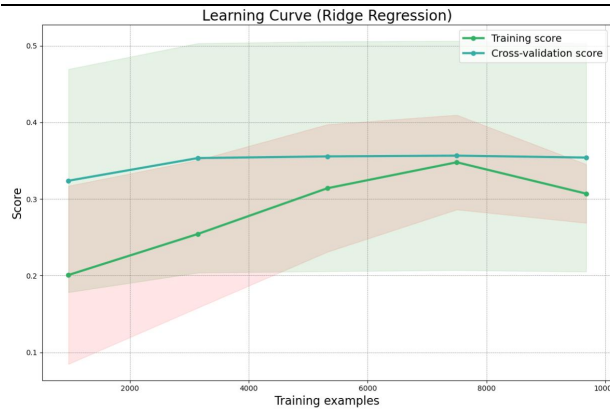


图 29: 岭回归训练过程

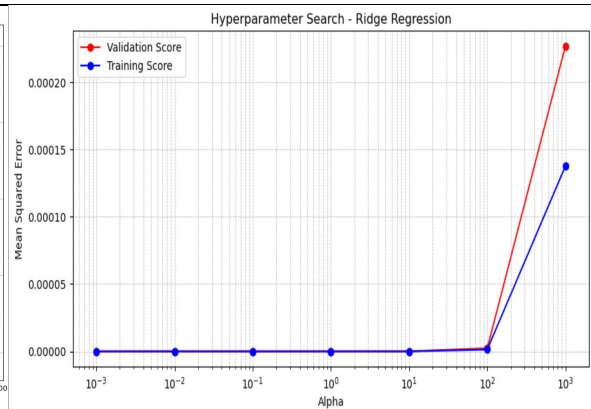


图 30: 岭回归超参数网格搜索过程

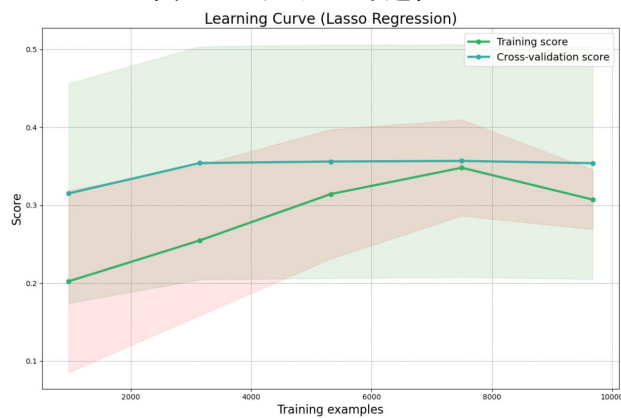


图 31: LASSO 回归训练过程

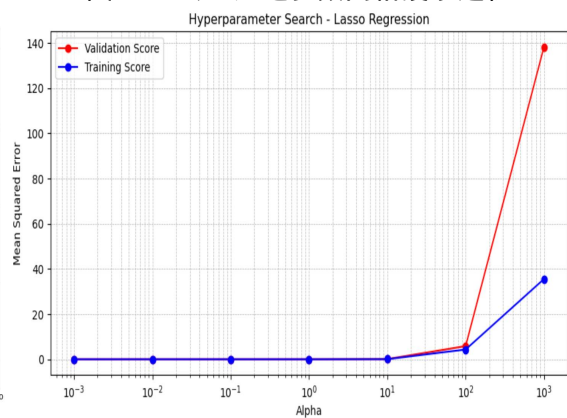


图 32: LASSO 回归超参数网格搜索过程

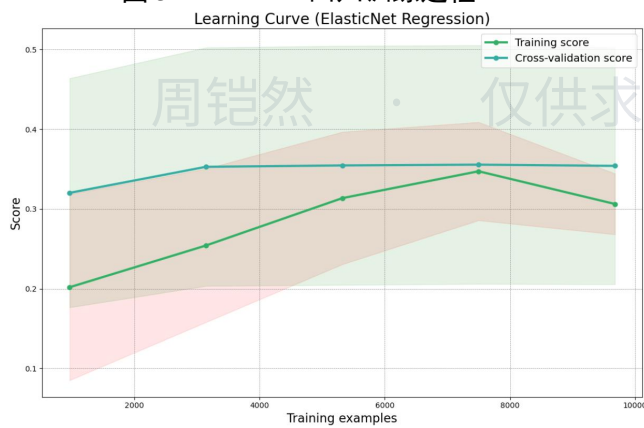


图 33: ElasticNet 回归训练过程

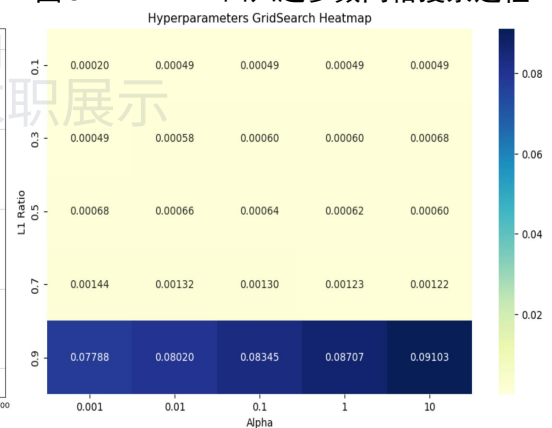


图 34: ElasticNet 回归超参数网格搜索过程

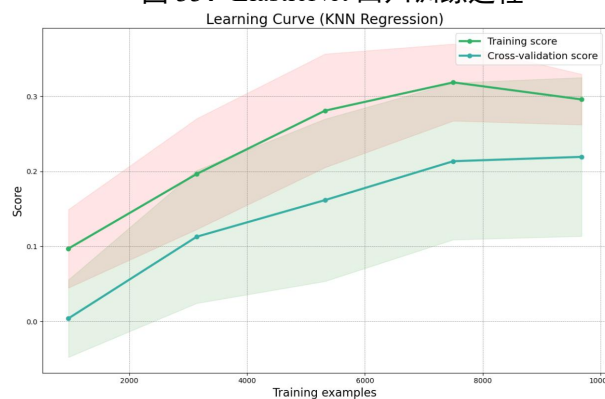


图 35: KNN 模型训练过程

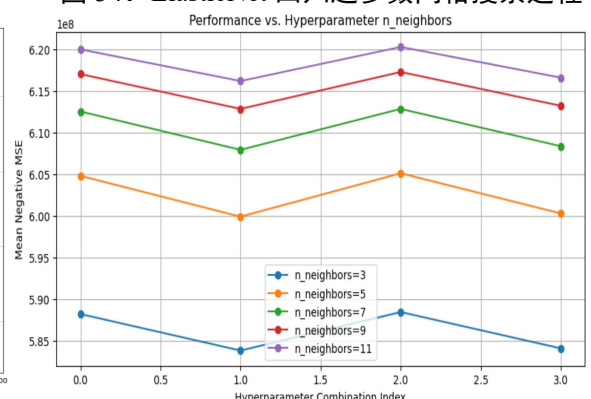


图 36: KNN 超参数网格搜索过程 (n_neighbors)

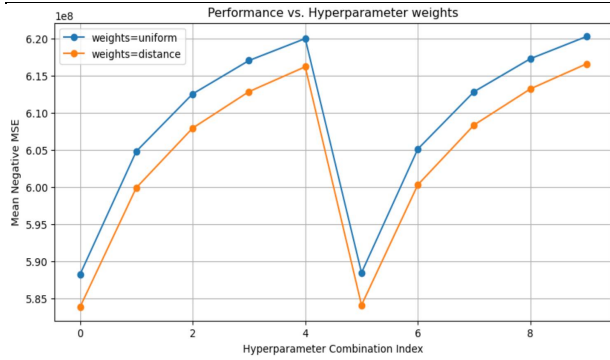


图 37: KNN 超参数网格搜索过程(weights)

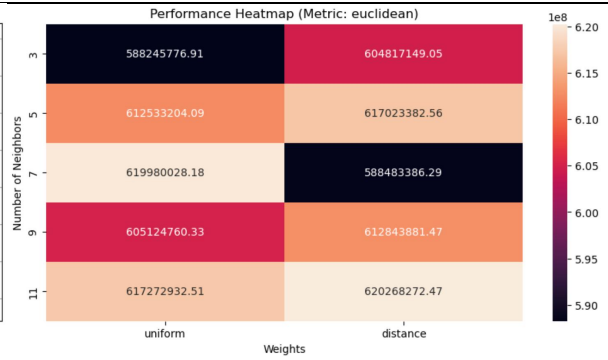


图 38: KNN 超参数组合网格搜索过程

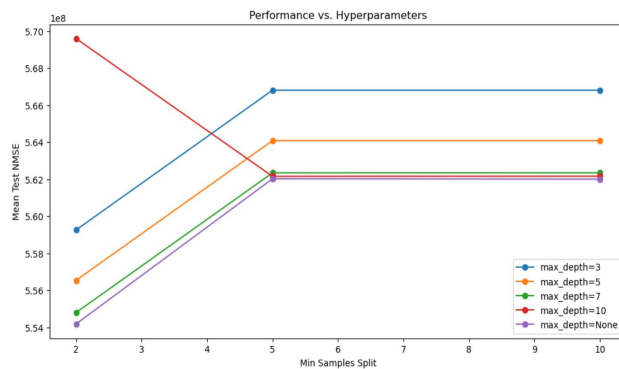


图 39: 决策树网格搜索过程

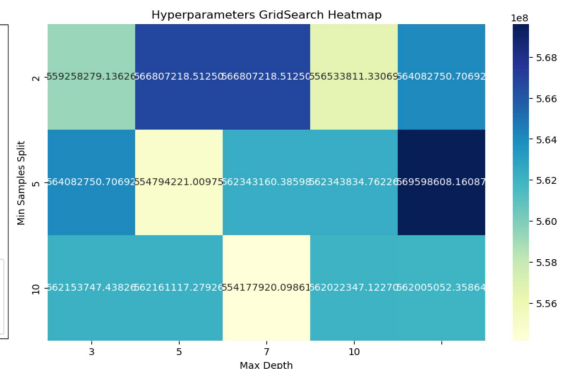


图 40: 决策树超参数组合网格搜索过程

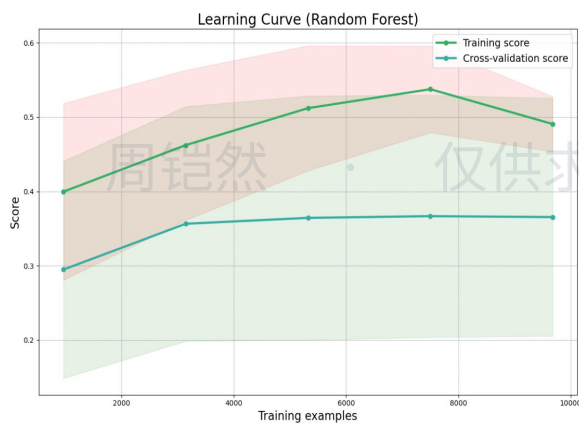


图 41: 随机森林训练过程

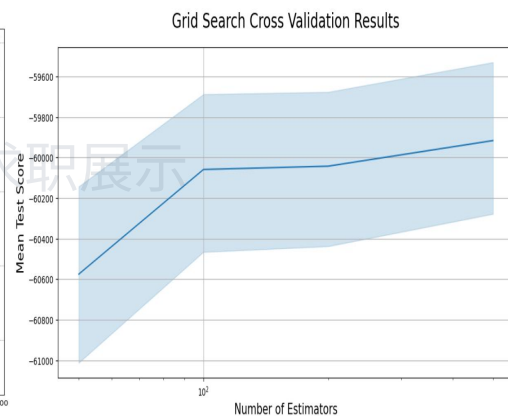


图 42: 随机森林超参数网格搜索过程

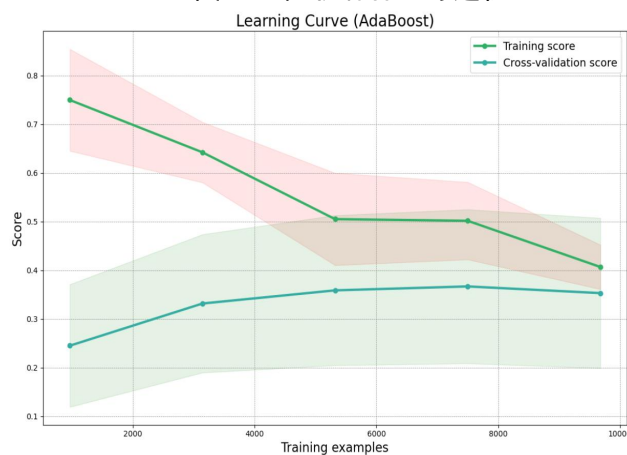


图 43: Adaboost 回归模型训练过程

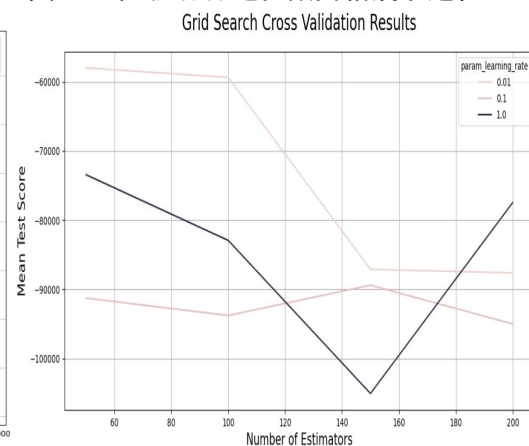


图 44: Adaboost 超参数网格优化

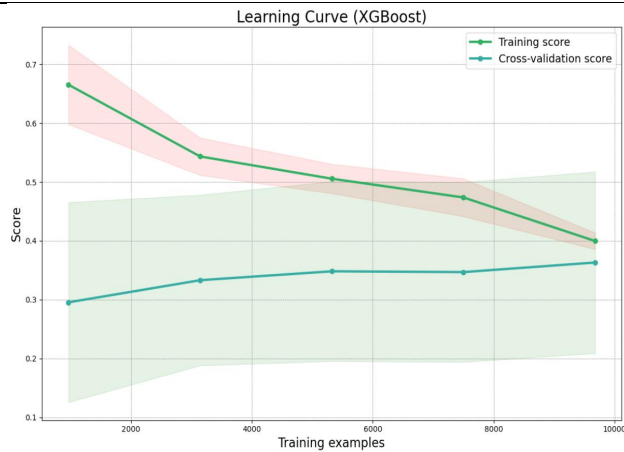


图 45: XGBoost 回归训练过程

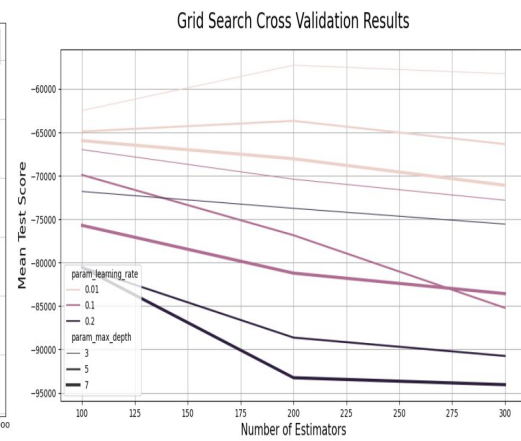


图 46: XGBoost 超参数网格搜索

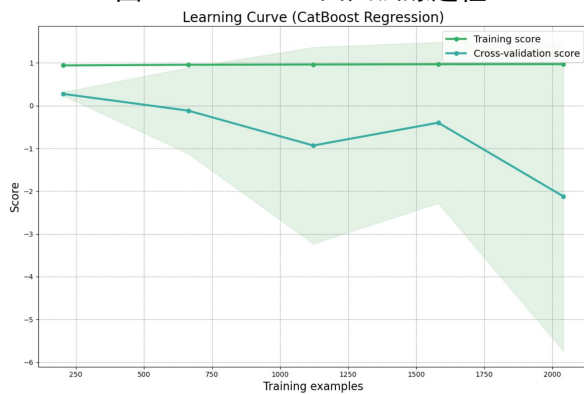


图 47: CatBoost 训练过程

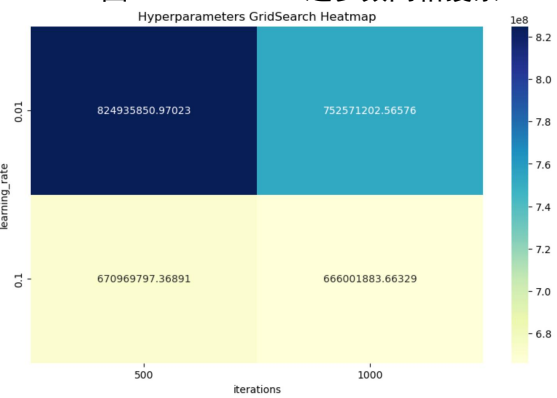


图 48: CatBoost 超参数组合网格搜索过程

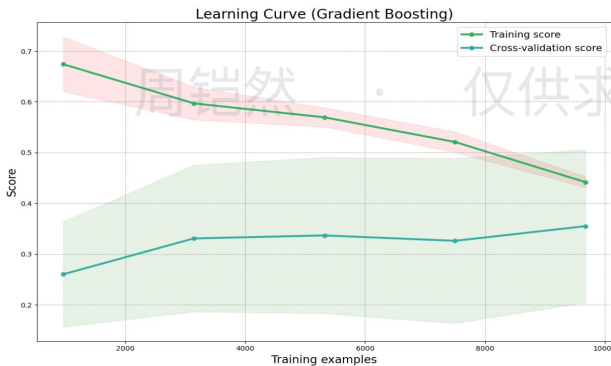


图 49: GradientBoosting 回归训练过程

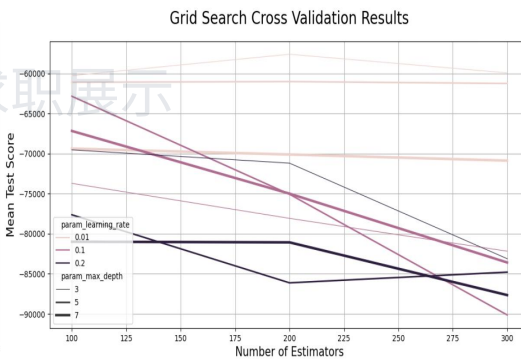


图 50: GradientBoosting 超参数网格搜索过程

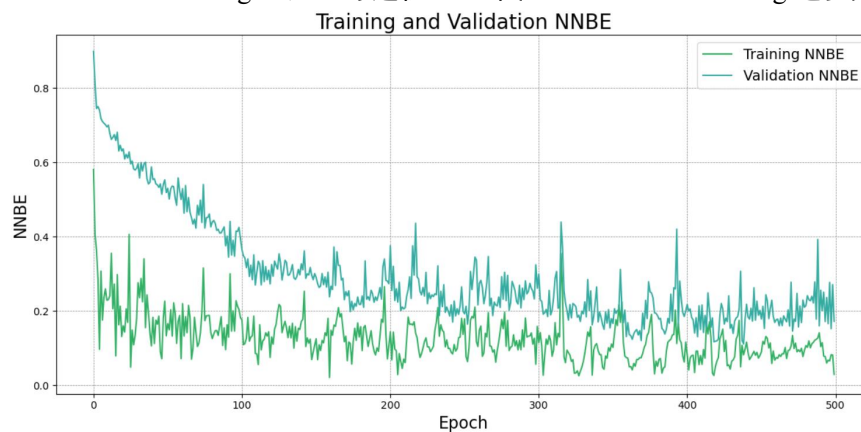


图 51: BP 神经网络训练过程

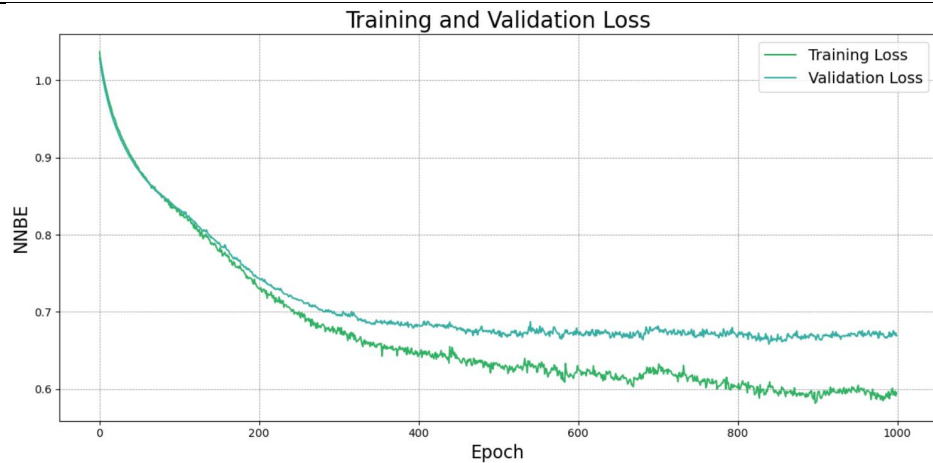


图 52: LSTM 训练过程

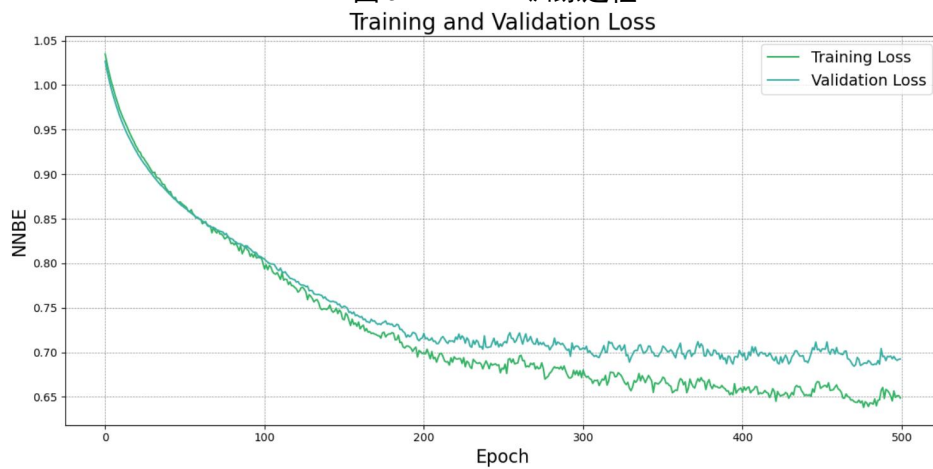


图 53: GRU 训练过程

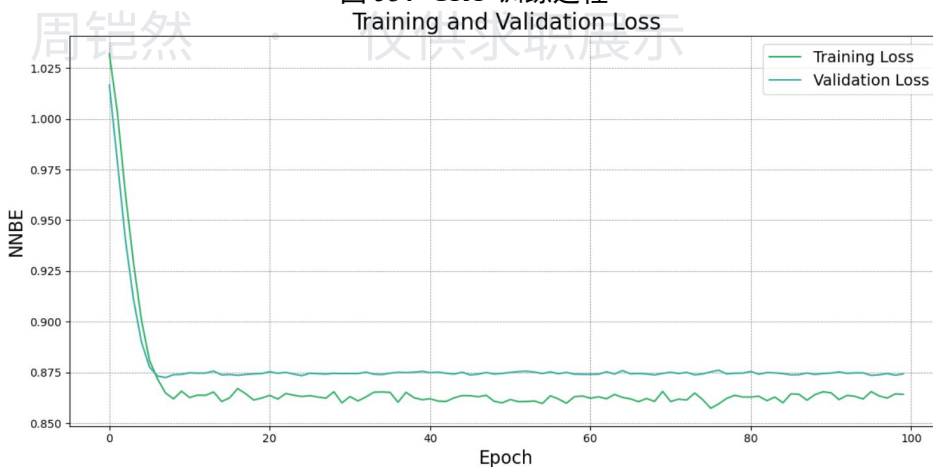


图 54: Transformer 训练过程

5.4 推荐系统预测结果展示

在本研究中，针对推荐系统的问题，采用了多种算法进行建模和分析，包括基于内容的推荐、协同过滤算法、奇异值分解(SVD)以及神经协同过滤(NCF)。

在对这些算法进行比较和分析后，神经协同过滤(NCF)在各项评价指标上均表现最为出色。具体各模型性能对比如表 6 所示。

表 6 各推荐系统效果对比表

Algorithm	Accuracy	MAE	Coverage	Novelty	Popularity Bias
基于内容的推荐	0.3263	384.9725	0.1982	0.0005	0.0528
协同过滤算法	0.0000	406.9114	0.2016	0.0017	0.0071
奇异值分解(SVD)	0.9544	293.7197	0.0212	29.7946	0.3950
神经协同过滤(NCF)	0.9762	65.6257	1.0000	0.5002	-0.1150

准确率表示模型预测正确的比例，近似 1 的值表明模型具有高度的预测准确性。MAE 则代表预测评分与实际评分之间的平均偏差，该模型的值为 65.6257，显示出其良好的预测效果。覆盖率为 1 意味着模型可以为每一个用户提供合适的推荐。新颖性为 0.5002，这一数值揭示了模型在推荐时综合了热门与新颖内容的权衡。流行度偏见的数值则测量模型是否偏向于推荐流行物品，近似 0 的值表示模型推荐的相对中立性。

为了更直观地展示模型的训练过程和性能，详见图 55 到图 58。

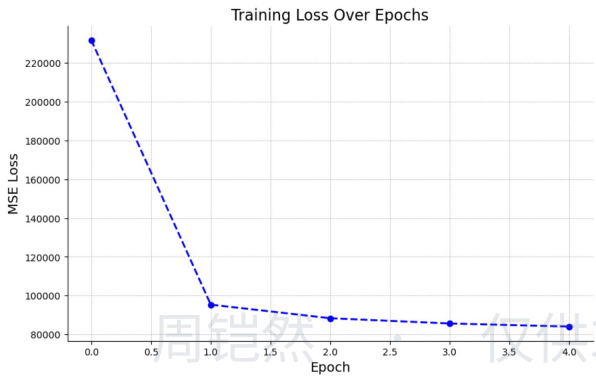


图 55：基于内容推荐模型训练过程

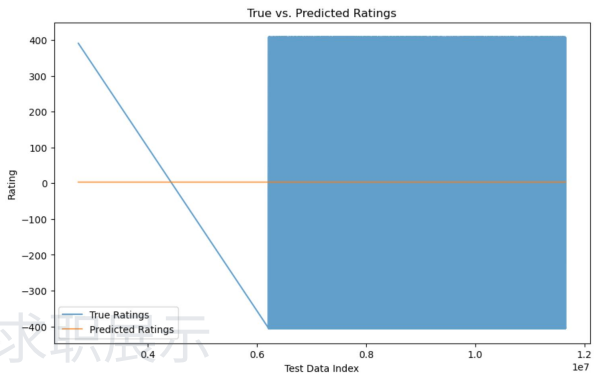


图 56：协同过滤模型训练过程

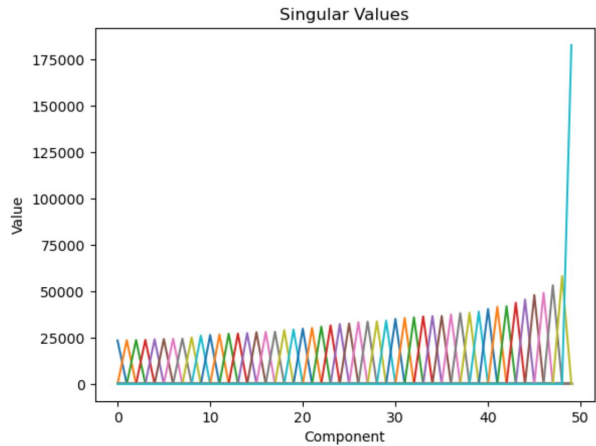


图 57：SVD 模型训练过程

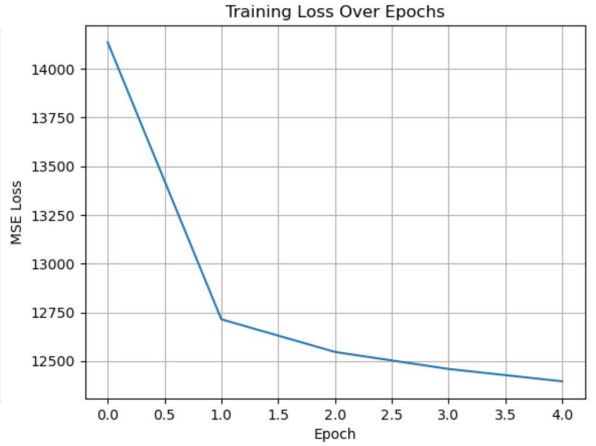


图 58：NCF 模型训练过程

第六章 讨论与深度分析

6.1 深度洞察与模型解读

6.1.1 智能手机用户行为模式解读

本研究运用了多种聚类算法，包括 K-Means、DBSCAN、Agglomerative Clustering 以及 Gaussian Mixture，对用户使用智能手机应用的行为进行了深度洞察。在定量评估的过程中，轮廓系数、Calinski-Harabasz 指数以及 Davies-Bouldin 指数等评价指标被采用，以衡量聚类结果的质量。在这些算法中，K-Means 算法表现最佳。这主要归因于 K-Means 算法的设计原理，即最小化每个聚类中的样本与其质心的距离，从而在众多实际问题中获得了有效的应用。

基于 K-Means 聚类的结果，构建了对应的用户画像，并对每个用户群体的行为特征进行了描述性统计。这些用户画像展示了用户在使用智能手机应用过程中的各种行为习惯和偏好。对于这些行为模式的深入分析，可以推测出用户可能的驱动因素，例如，个人兴趣、工作需求或社交影响等。这些洞察可能有助于更好地理解用户行为，进而对个性化推荐系统的发展和用户体验优化提供重要的指导。

6.1.2 用户应用使用情况预测分析

用户应用使用情况的预测是本研究的另一重要内容。在解决分类问题上，本研究尝试了多种机器学习和深度学习算法，包括逻辑回归、K-近邻（KNN）、朴素贝叶斯、随机森林、梯度提升树（GBDT）、XGBoost、AdaBoost、BP 神经网络以及卷积神经网络（CNN）等。结果显示，AdaBoost 模型在分类问题上的表现最优，准确率高达 0.9968，并且 Hamming Loss 为 0.000152。这一优异表现主要来自于 AdaBoost 的核心机制。AdaBoost 是一种自适应的迭代算法，能够针对之前分类错误的样本给予更高的权重，使得这些样本在后续的训练中得到更多的关注。这使得在实际用户 APP 使用数据中可能存在的样本不均衡问题得到了有效的处理。同时，由于每次迭代都是针对之前的错误进行，弱分类器的多样性使得组合后的强分类器更加鲁棒，更好地泛化到未知数据。再者，AdaBoost 的权重调整机制有助于避免模型过拟合，与一些复杂的模型相比，在测试集上往往表现更好。最后，该算法在训练过程中会自动进行特征选择，这意味着一些关键的、可以更好地预测用户行为的特征会被模型捕捉。

在回归问题上，本研究主要关注了用户使用应用的时长预测。在应用了多元线性回归、岭回归、LASSO 回归、弹性网络回归（ElasticNet）、K-近邻回归（KNN）、随机森林回归、梯度提升树回归（GBDT）、XGBoost 回归、AdaBoost 回归、BP 神经网络回归、CNN、LSTM、GRU 等一系列回归模型后，结果表明，BP 神经网络在预测用户

应用使用时长上取得了最好的效果，其标准化均方误差（NMSE）为 0.1725。这一优异表现主要来自于 BP 神经网络的特有算法结构。BP 神经网络是一种基于反向传播算法的多层前馈网络，它通过逐层计算误差并向前传播以更新权重。这种逐步优化的策略使得模型能够系统地减少误差，特别是在存在非线性和高维特征交互的数据集中表现尤为出色。此外，BP 神经网络的多层结构使其具有强大的表示能力，能够捕获更复杂的数据模式。当处理复杂的应用使用时长数据时，它可以捕获更多潜在的模式和关系，从而更准确地预测用户的应用使用时长。

在推荐系统构建中，本研究主要关注了用户对 APP 的使用偏好预测。在尝试了基于内容的推荐、协同过滤算法、奇异值分解（SVD）以及神经协同过滤（NCF）等一系列推荐算法后，结果表明，神经协同过滤(NCF)在预测用户对 APP 的使用偏好上取得了最好的效果，其准确率高达 0.9762。这一优异表现主要来自于 NCF 的特有算法结构。

神经协同过滤结合了传统的矩阵分解技术和多层感知机的能力，使得模型能够学习到用户和项目之间的非线性关系。在 NCF 中，用户和项目的隐向量不仅仅是简单地相乘，而是被输入到一个多层感知机中，从而捕获更加复杂的交互模式。这种结构允许模型深入挖掘用户的潜在兴趣和偏好，从而大大提高了预测的准确性。

此外，NCF 在训练过程中，通过使用负采样技术，有效地解决了正负样本不平衡的问题，进一步提高了模型的泛化能力。并且，通过使用深度学习技术，NCF 能够自动识别并重视对预测最有贡献的特征。这意味着在复杂的用户使用偏好数据中，它可以捕获更多潜在的模式和关系，从而更准确地预测用户对 APP 的使用偏好。

总的来说，本研究在不同的预测任务上均获得了令人满意的结果。无论是用户应用使用情况的分类预测、使用时长的回归预测还是对 APP 的使用偏好推荐，都表明了选择合适的模型和算法至关重要。AdaBoost、GradientBoosting 和神经协同过滤分别在各自的领域中展现了其独特的优势，为更加精准地理解和预测用户行为提供了有力的工具。这为后续的应用和优化提供了宝贵的经验，有助于进一步提升模型的实际应用价值。

6.2 研究限制与潜在改进

本研究对智能手机用户的监测数据进行了深入剖析，以揭示用户行为模式和应用使用习惯的特征。然而，如同任何科学探索，本研究中也存在一些局限性，并有待于未来研究的进一步改进。

对于数据预处理环节，尽管已经实施了标准化和标签编码等预处理步骤，但这些方法可能无法完全揭示数据的复杂性。例如，线性标准化可能无法完全捕获具有非线性分布特性的特征，而标签编码可能在处理分类特征时引入了不必要的序关系。未来的研究可以考虑引入更复杂的预处理方法，例如非线性变换、独热编码或者目标编码等，以更精确地揭示数据的内在结构。

在大数据处理方面，尽管已经采用了如 **Vaex** 库、批处理和 **GPU** 加速等高效的数据处理技术，但在处理更大规模的数据时，可能需要引入更为强大的分布式计算框架，如 **Apache Spark** 或者 **Dask** 等。这不仅能提高数据处理的效率，还可以支持更大规模的数据分析任务。

在进行聚类分析时，虽然本研究应用了多种聚类算法并进行了比较，但对聚类结果的解读以及用户画像的构建存在一定的主观性，可能影响到结果的公正性和可靠性。未来的研究可以考虑引入更为严谨的聚类结果评估方法，或者使用更为客观的用户画像构建框架，以提高结果的客观性和可信度。

在预测模型的应用中，本研究使用了多种机器学习和深度学习模型，并采用了交叉验证以及 **NMSE** 等评估指标。然而，模型的预测性能可能受到模型复杂度、数据质量和数量、超参数选择等多种因素的影响。为了提高预测的准确性和模型的泛化能力，未来的研究可以尝试采用更为复杂的模型，如深度神经网络或者强化学习模型，或者通过更为精细的超参数优化和模型正则化技术来防止过拟合。

在推荐系统的构建中，本研究采用了多种推荐算法，如基于内容的推荐、协同过滤算法、奇异值分解和神经协同过滤等，并评估了这些算法的预测性能。然而，推荐系统仍有一些未解决的挑战和局限性。例如，冷启动问题是推荐系统中的一个经典问题，即如何为新用户或新产品提供准确的推荐。此外，模型可能会受到数据稀疏性的影响，导致预测不准确。未来的研究可以考虑引入基于知识图谱的推荐方法，或者深度强化学习来解决这些挑战。还有，推荐系统中的算法偏见和不公平性也是一个重要的问题，未来的研究可以考虑引入公平性和可解释性技术，确保推荐结果的公正性和透明性。

总的来说，无论是用户行为预测、用户使用时长预测还是推荐系统，本研究都为智能手机用户的行为分析提供了一个全面的框架。但在每一块领域中，都还存在一些可以进一步探索和优化的地方。期望通过持续的研究和技术进步，能够为智能手机用户行为的更深入、更准确的分析和预测提供更加完善的解决方案。

第七章 结论与未来研究方向

7.1 研究成果与贡献

本研究在深入探讨了 2023 年第二届钉钉杯大学生大数据挑战赛的主题——智能手机用户监测数据分析的过程中，实现了用户行为的深度解析，构建出精确的用户画像，并对用户未来 APP 使用行为进行了准确预测。这些研究结果在理论层面为用户行为研究提供了新的视角，同时在实际操作中为 APP 产品优化以及个性化服务提供了技术保障。

首先，本研究通过用户常用的 20 类 APP 的数据，实现了对用户行为模式的深度挖掘。利用了高级的聚类算法，揭示了用户群体的内在行为模式和 APP 使用偏好。这一结果不仅有助于理解用户的行为习惯，而且为定制化的 APP 推荐提供了实用的依据。

其次，通过对聚类结果的深度分析，本研究成功构建了用户画像，并针对不同用户群体的特性进行了深入剖析。这些用户画像全面描绘了用户的属性、偏好和行为习惯，为更精确地理解用户需求，进一步推动 APP 产品的优化和个性化服务的提供，奠定了坚实的基础。

进一步地，本研究通过应用一系列的机器学习和深度学习模型，本研究实现了对用户未来是否会使用 APP 及其使用时长的预测。这些预测模型不仅有助于理解用户的 APP 使用行为，也为预测用户行为和 APP 推荐提供了重要的技术支持。

最后，在推荐系统构建中，主要关注了用户对 APP 的使用偏好预测。采用了多种推荐算法，如基于内容的推荐、协同过滤算法、奇异值分解和神经协同过滤等，对各种算法的预测性能进行了深入的评估和比较。尤其是神经协同过滤模型在预测用户对 APP 的使用偏好上展现出了卓越的性能，这一部分的研究结果进一步完善了对用户行为的全方位分析，为个性化的 APP 推荐提供了强大的技术支持。

综合来看，本研究的主要贡献在于将大数据分析、先进的机器学习技术和推荐系统的构建相结合，应用于智能手机用户行为的分析和预测中，从而为理解用户行为、优化 APP 产品、提供个性化服务等提供了重要的理论和技术支持。同时，本研究也展示了如何利用高效的数据处理技术来处理大规模的智能手机用户监测数据，为大数据分析提供了有力的技术支持。这一全面的研究成果，无疑为智能手机用户行为研究的进展，提供了重要的理论和实践指导。

7.2 未来研究方向展望

未来的研究方向可从以下几个维度进行拓展与深化：

1.模型算法的优化与创新：虽然当前的机器学习与深度学习模型已取得一定成果，但仍存在诸多局限，尤其是在处理非线性、高维、稀疏数据时。未来可以考虑结合图神经网络(GNN)等先进模型，以更好地挖掘数据之间的隐含结构与关系。此外，模型的集成、迁移学习以及元学习等策略也是值得探索的领域。

2.隐私保护与数据安全：在智能手机用户数据分析中，隐私问题日益受到关注。未来研究需要重点考虑如何在确保数据隐私的前提下进行高效分析。差分隐私、同态加密以及联邦学习等技术可以为此提供技术支持和解决方案。

3.上下文信息的整合：用户行为数据往往受到多种上下文因素的影响，如时间、地点、设备、外部环境等。在模型构建时，如何有效地融合这些上下文信息，以增强模型的预测能力，是一个重要方向。

4.多模态数据分析：除了传统的行为数据，用户还会产生声音、图像、视频等多种类型的数据。如何有效地进行多模态数据的融合与分析，以获取更全面的用户画像和行为模式，是一个挑战性的问题。

5.大数据计算与处理：随着数据规模的快速增长，单机的计算能力已难以满足分析需求。未来的研究可以考虑更为高效的分布式计算框架，如 Apache Spark，以及更先进的数据存储与查询技术，满足大数据分析的实时性与高效性需求。

综上所述，智能手机用户行为研究仍有诸多待挖掘的领域和方向。这些方向不仅有望进一步提升用户行为预测的准确性，还能够为行业实践提供更为深入、系统的理论指导与技术支持。

参考文献

- [1] 刘友田, 张泽. 智能传播时代网络空间安全治理的价值逻辑, 现实困境与路径选择[J]. 观察与思考, 2023 (3): 88-96.
- [2] 韩志明, 刘子扬. 穿透稠密的城市空间——数字时代超大规模城市的清晰化之道[J]. 《广西师范大学学报》(哲学社会科学版), 2022, 58(3): 23-36.
- [3] 毛建军. 基于改进 K-Means 聚类算法的移动 5G 手机用户分类研究[J]. 网络安全技术与应用, 2021, No.242(02): 40-41.
- [4] 黄立威, 江碧涛, 吕守业, et al. 基于深度学习的推荐系统研究综述[J]. 计算机学报, 2018, 41(07): 1619-1647.
- [5] 童婉迪. 基于优化的 K-means 算法的电信业手机用户细分及维护策略研究[D]. 安徽大学, 2021.
- [6] 韩迪, 李雯婷, 王庆娟等. 增量学习的优化算法在 app 使用预测中的应用[J]. 深圳大学学报(理工版), 2019, 36(01): 43-51.
- [7] 蒋孜文. AppUsage2Vec: 基于手机 App 使用记录的用户行为建模与应用[D]. 浙江大学, 2018.
- [8] Wu L, He X, Wang X, et al. A Survey on Neural Recommendation: From Collaborative Filtering to Content and Context Enriched Recommendation[J]. arXiv, 2021, abs/2109.00000.
- [9] Xin H, Jiameng B, Jia P, et al. Content-based Re-ranking Method for Recommendation[J]. IOP Conference Series: Materials Science and Engineering, 2020, 799(1).
- [10] 李宏志, 李苑兰. 融合改进的内容与协同过滤的博客推荐方法[J]. 湖南科技大学学报(自然科学版), 2021, 36(03): 104-112.
- [11] Francesco C, Dajana C, Massimo S D, et al. A content-based recommendation approach based on singular value decomposition[J]. Connection Science, 2022, 34(1).
- [12] Sridhar S, Dhanasekaran D, Latha P C G. Content-Based Movie Recommendation System Using MBO with DBN[J]. Intelligent Automation & Soft Computing, 2023, 35(3).
- [13] 冯萍, 钱阳, 李国梁等. 融合图神经网络的深度学习电影推荐系统设计与实现[J]. 白城师范学院学报, 2021, 35(05): 49-56.
- [14] Bifeng L, Gangfeng L, Jingxiu X, et al. A personalized recommendation framework based on MOOC system integrating deep learning and big data[J]. Computers and Electrical Engineering, 2023, 106.
- [15] 卫刚, 邵伟, 王志成. 新闻推荐系统中的边信息融合卷积神经网络[J]. 同济大学学报(自然科学版), 2022, 50(04): 590-600.
- [16] 王国霞, 刘贺平. 个性化推荐系统综述[J]. 计算机工程与应用, 2012: 66-76.
- [17] Jain, A. K., Murty, M. N., & Flynn, P. J. (1999). Data clustering: A review. ACM Computing Surveys (CSUR), 31(3), 264-323.
- [18] Steinley, D. (2004). Properties of the Hubert-Arabie adjusted Rand index. Psychological methods, 9(3), 386-396.

-
- [19] Xie, X. L., & Beni, G. (1991). A validity measure for fuzzy clustering. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(8), 841-847.
- [20] Hastie, T., Tibshirani, R., & Friedman, J. (2009). *The elements of statistical learning: Data mining, inference, and prediction* (2nd ed.). Springer.
- [21] Shalev-Shwartz, S., & Ben-David, S. (2014). *Understanding machine learning: From theory to algorithms*. Cambridge University Press.
- [22] James, G., Witten, D., Hastie, T., & Tibshirani, R. (2013). *An introduction to statistical learning: With applications in R*. Springer.
- [23] Box, G. E., Jenkins, G. M., Reinsel, G. C., & Ljung, G. M. (2015). *Time series analysis: Forecasting and control* (5th ed.). Wiley.
- [24] Krizhevsky, A., Sutskever, I., & Hinton, G. E. ImageNet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 2012, 25: 1097-1105.
- [25] Simonyan, K., & Zisserman, A. Very deep convolutional networks for large-scale image recognition. *International Conference on Learning Representations (ICLR)*, 2015.
- [26] Szegedy, C., et al. Going deeper with convolutions. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015: 1-9.
- [27] Ioffe, S., & Szegedy, C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *International Conference on Machine Learning (ICML)*, 2015: 448-456.
- [28] He, K., Zhang, X., Ren, S., & Sun, J. Deep residual learning for image recognition. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016: 770-778.
- [29] Girshick, R., et al. Region-based convolutional networks for accurate object detection and segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 2016, 38(1): 142-158.
- [30] Huang, D.-A., et al. Attention-based models for speech recognition. *Advances in Neural Information Processing Systems 29 (NIPS)*, 2016: 577-585.
- [31] Vaswani, A., et al. Attention is all you need. *Advances in Neural Information Processing Systems 30 (NIPS)*, 2017: 5998-6008.
- [32] Zhu, Y., et al. DeeCampus: Campus-wide WiFi localization via deep neural network. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies (IMWUT)*, 2017, 1(3): Article No. 58.

附录

硬件设备：CPU:AMD Ryzen 9 7945HX with Radeon Graphics 2.50 GHz; GPU:RTX4060

软件名称：Jupyter Notebook

程序源代码：

“=====读取数据=====”

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from glob import glob
from sklearn.preprocessing import StandardScaler
import warnings
warnings.filterwarnings("ignore")

# 定义路径
app_class_path = "C:/Users/Desktop/A/app_class.csv"
days_data_path = "C:/Users/Desktop/A/day*.txt"

# 读取 app 类别数据
app_class = pd.read_csv(app_class_path, header=None, names=["appid", "app_class"], dtype={"appid": str,
"app_class": str})

# 读取每日用户数据
user_data_files = glob(days_data_path)

# 为了节省内存，我们将读取的数据类型进行转换
dtypes = {
    "uid": str,
    "appid": str,
    "app_type": str,
    "start_day": np.int8,
    "start_time": str,
    "end_day": np.int8,
    "end_time": str,
    "duration": np.int32,
    "up_flow": np.int32,
    "down_flow": np.int32,
}

# 读取并合并数据
data = pd.DataFrame()
for file in user_data_files:
    day_data = pd.read_csv(file, header=None, names=["uid", "appid", "app_type", "start_day",
"start_time",
"end_day", "end_time", "duration",
"up_flow", "down_flow"], dtype=dtypes)
    data = pd.concat([data, day_data], ignore_index=True)

# 与 app_class 进行合并
```

```
data = pd.merge(data, app_class, on="appid", how="left")
data.info()
```

```
“=====数据预处理=====”
```

```
# 剔除 duration、up_flow 和 down_flow 为 0 的行
data = data[(data["duration"] > 0) & (data["up_flow"] > 0) & (data["down_flow"] > 0)]

# 剔除使用时长过于短（小于 5 秒）的行
data = data[data["duration"] >= 5]

# 将使用时长的单位转换为分钟
data["duration"] = data["duration"] / 60

# 将流量单位转换为 MB
data["up_flow"] = data["up_flow"] / (1024 * 1024)
data["down_flow"] = data["down_flow"] / (1024 * 1024)

# 对于未知类别的 APP，我们将其类别标记为"unknown"
data["app_class"] = data["app_class"].fillna("unknown")

# 在数据标准化之前，取出"duration", "up_flow", "down_flow"的部分数据进行可视化
data_before_scaling = data[["duration", "up_flow", "down_flow"]].copy(deep=True)

# 对"duration", "up_flow", "down_flow"这三列数据进行标准化处理
scaler = StandardScaler()
data[["duration", "up_flow", "down_flow"]] = scaler.fit_transform(data[["duration", "up_flow",
"down_flow"]])

# 展示处理后的数据
data.head()
```

周铠然 · 仅供求职展示

```
“=====特征工程=====”
```

```
# 导入必要的库
from datetime import datetime, timedelta

# 定义一个函数用于转换 start_day 和 end_day 为日期格式，并计算使用 APP 的时间跨度
def get_date_span(row):
    # 假设数据开始的日期为 2023 年 8 月 1 日
    start_date = datetime(2023, 8, 1)
    # 计算起始和结束日期
    row["start_date"] = start_date + timedelta(days=row["start_day"] - 1)
    row["end_date"] = start_date + timedelta(days=row["end_day"] - 1)
    # 计算使用 APP 的时间跨度（单位为天）
    row["date_span"] = (row["end_date"] - row["start_date"]).days + 1
    return row

# 对数据进行处理
data = data.apply(get_date_span, axis=1)

# 对"uid"和"appid"进行分组，计算每个用户每个 APP 的使用总时长、总流量、平均每次使用时长、
# 平均流量
grouped_user_app = data.groupby(["uid", "appid"]).agg({
    "duration": ["sum", "mean"],
    "up_flow": ["sum", "mean"],
```

```

        "down_flow": ["sum", "mean"],
        "start_date": "min",
        "end_date": "max",
        "date_span": "max"
    }).reset_index()
grouped_user_app.columns = ["uid", "appid", "total_duration", "average_duration", "total_up_flow",
"average_up_flow",
                                "total down flow", "average down flow", "first use date",
"last_use_date", "date_span"]

# 对"uid"进行分组，计算每个用户的使用 APP 数量、使用总时长、总流量、平均每次使用时长、平
均流量
grouped_user = data.groupby("uid").agg({
    "appid": "nunique",
    "duration": ["sum", "mean"],
    "up_flow": ["sum", "mean"],
    "down_flow": ["sum", "mean"],
    "start_date": "min",
    "end_date": "max",
    "date_span": "max"
}).reset_index()
grouped_user.columns = ["uid", "app count", "total duration", "average duration", "total up flow",
"average_up_flow",
                                "total_down_flow", "average_down_flow", "first_use_date", "last_use_date",
"date_span"]

# 对"appid"进行分组，计算每个 APP 被使用的次数、总时长、总流量、平均每次使用时长、平均流
量
grouped_app = data.groupby("appid").agg({
    "uid": "nunique",
    "duration": ["sum", "mean"],
    "up_flow": ["sum", "mean"],
    "down_flow": ["sum", "mean"],
    "start_date": "min",
    "end_date": "max",
    "date_span": "max"
}).reset_index()
grouped_app.columns = ["appid", "user_count", "total_duration", "average_duration", "total_up_flow",
"average up flow",
                                "total down flow", "average down flow", "first use date", "last use date",
"date_span"]

# 展示处理后的数据
grouped_user_app.head(), grouped_user.head(), grouped_app.head()

import seaborn as sns

# 设置 Seaborn 风格
sns.set(style="whitegrid")

# 查看每个用户使用的 APP 数量的分布
plt.figure(figsize=(10, 6))
sns.histplot(grouped_user['app count'], bins=30, color='skyblue', kde=True)
plt.title('Distribution of the number of apps used by each user', fontsize=15)
plt.xlabel('Number of apps', fontsize=13)
plt.ylabel('Count', fontsize=13)
plt.show()

```

```

# 查看每个 APP 被使用的次数的分布
plt.figure(figsize=(10, 6))
sns.histplot(grouped_app['user_count'], bins=30, color='skyblue', kde=True)
plt.title('Distribution of the number of times each app is used', fontsize=15)
plt.xlabel('Number of users', fontsize=13)
plt.ylabel('Count', fontsize=13)
plt.show()

# 查看每个用户的使用总时长和总流量的关系
plt.figure(figsize=(10, 6))
sns.scatterplot(x='total_duration', y='total_up_flow', data=grouped_user, alpha=0.5, color='g')
plt.title('Relationship between total duration and total flow for each user', fontsize=15)
plt.xlabel('Total Duration', fontsize=13)
plt.ylabel('Total Flow', fontsize=13)
plt.show()

# 查看每个 APP 的总时长和总流量的关系
plt.figure(figsize=(10, 6))
sns.scatterplot(x='total_duration', y='total_up_flow', data=grouped_app, alpha=0.5, color='purple')
plt.title('Relationship between total duration and total flow for each app', fontsize=15)
plt.xlabel('Total Duration', fontsize=13)
plt.ylabel('Total Flow', fontsize=13)
plt.show()

# 对"uid"和"start_day"进行分组，计算每个用户每天的使用 APP 数量、使用时长、流量
grouped_user_day = data.groupby(["uid", "start_day"]).agg({
    "appid": "nunique",
    "duration": ["sum", "mean"],
    "up_flow": ["sum", "mean"],
    "down_flow": ["sum", "mean"],
}).reset_index()
grouped_user_day.columns = ["uid", "start_day", "app_count", "total_duration", "average_duration",
                           "total_up_flow",
                           "average_up_flow", "total_down_flow", "average_down_flow"]

# 对"appid"和"start_day"进行分组，计算每个 APP 每天被使用的次数、使用时长、流量
grouped_app_day = data.groupby(["appid", "start_day"]).agg({
    "uid": "nunique",
    "duration": ["sum", "mean"],
    "up_flow": ["sum", "mean"],
    "down_flow": ["sum", "mean"],
}).reset_index()
grouped_app_day.columns = ["appid", "start_day", "user_count", "total_duration", "average_duration",
                           "total_up_flow",
                           "average_up_flow", "total_down_flow", "average_down_flow"]

# 计算用户使用 APP 的总天数
user_day_count = data.groupby("uid")["start_day"].nunique().reset_index()
user_day_count.columns = ["uid", "day_count"]

# 计算每种 APP 被使用的天数
app_day_count = data.groupby("appid")["start_day"].nunique().reset_index()
app_day_count.columns = ["appid", "day_count"]

# 对用户的"total_duration"特征进行分箱处理

```

```

grouped_user["total_duration_bin"] = pd.qcut(grouped_user["total_duration"], q=10, duplicates='drop')

# 对 APP 的"total_duration"特征进行分箱处理
grouped_app["total_duration_bin"] = pd.qcut(grouped_app["total_duration"], q=10, duplicates='drop')

import seaborn as sns
import matplotlib.pyplot as plt

sns.set(style="whitegrid") # 设置 Seaborn 风格

# 查看每个用户每天的使用 APP 数量的分布
plt.figure(figsize=(10, 6))
sns.histplot(grouped_user_day['app_count'], bins=30, color='skyblue', kde=True)
plt.title('Distribution of the number of apps used by each user per day', fontsize=15)
plt.xlabel('Number of apps', fontsize=13)
plt.ylabel('Count', fontsize=13)
plt.show()

# 查看每个 APP 每天被使用的次数的分布
plt.figure(figsize=(10, 6))
sns.histplot(grouped_app_day['user count'], bins=30, color='skyblue', kde=True)
plt.title('Distribution of the number of times each app is used per day', fontsize=15)
plt.xlabel('Number of users', fontsize=13)
plt.ylabel('Count', fontsize=13)
plt.show()

# 查看用户使用 APP 的总天数的分布
plt.figure(figsize=(10, 6))
sns.histplot(user_day_count['day_count'], bins=30, color='skyblue', kde=True)
plt.title('Distribution of the total number of days each user uses apps', fontsize=15)
plt.xlabel('Number of days', fontsize=13)
plt.ylabel('Count', fontsize=13)
plt.show()

# 查看每种 APP 被使用的天数的分布
plt.figure(figsize=(10, 6))
sns.histplot(app_day_count['day_count'], bins=30, color='skyblue', kde=True)
plt.title('Distribution of the total number of days each app is used', fontsize=15)
plt.xlabel('Number of days', fontsize=13)
plt.ylabel('Count', fontsize=13)
plt.show()

# 查看用户总使用时长的分箱分布
plt.figure(figsize=(10, 6))
sns.countplot(x='total_duration_bin', data=grouped_user, palette='viridis')
plt.title('Distribution of total usage duration (binned) for each user', fontsize=15)
plt.xlabel('Total Duration (binned)', fontsize=13)
plt.ylabel('Count', fontsize=13)
plt.xticks(rotation=45)
plt.show()

# 查看 APP 总使用时长的分箱分布
plt.figure(figsize=(10, 6))
sns.countplot(x='total_duration_bin', data=grouped_app, palette='viridis')
plt.title('Distribution of total usage duration (binned) for each app', fontsize=15)
plt.xlabel('Total Duration (binned)', fontsize=13)

```

```
plt.ylabel('Count', fontsize=13)
plt.xticks(rotation=45)
plt.show()
```

“=====聚类分析=====”

```
import pandas as pd
import numpy as np
from sklearn.cluster import KMeans, DBSCAN, AgglomerativeClustering
from sklearn.mixture import GaussianMixture
from sklearn.metrics import accuracy_score, mean_squared_error
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings("ignore")
%matplotlib inline

import vaex
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler

# Read app_class file
app_class = vaex.from_csv("C:\\Users\\[redacted]\\Desktop\\A\\app_class.csv", header=None, names=['appid',
'app_class'])

# Convert appid to string to ensure consistent data types
app_class['appid'] = app_class['appid'].apply(lambda x: str(x))

# Create empty DataFrame to store results
grouped_data = None

# Process data
for i in range(1, 31):
    day = str(i).zfill(2)
    file_path = f"C:\\Users\\[redacted]\\Desktop\\A\\day{day}.txt"
    data = vaex.from_csv(file_path, header=None, names=['uid', 'appid', 'app_type', 'start_day', 'start_time',
'end_day', 'end_time', 'duration', 'up_flow', 'down_flow'])

    # Convert appid to string to ensure consistent data types
    data['appid'] = data['appid'].apply(lambda x: str(x))

    # Merge with app_class
    data = data.join(app_class, on='appid', how='left', allow_duplication=True) # Added
allow_duplication=True

    # Handle missing values for numeric columns only
    numeric_cols = ['duration', 'up_flow', 'down_flow']
    for col in numeric_cols:
        data[col] = data[col].fillna(data[col].mean())

    # Feature selection
    data = data[['uid', 'app_class', 'duration', 'up_flow', 'down_flow']]

    # Compute mean duration, total up_flow and down_flow for each user and each app class
    grouped = data.groupby(['uid', 'app_class']).agg({'duration': 'mean', 'up_flow': 'sum', 'down_flow':
'sum'})

    # Append to grouped_data
```

```

    if grouped_data is None:
        grouped_data = grouped
    else:
        grouped_data = grouped_data.concat(grouped)

# Convert Vaex DataFrame to Pandas DataFrame for pivot operation
grouped_data = grouped_data.to_pandas_df()

# Construct feature matrix
features = grouped_data.pivot_table(index='uid', columns='app_class', values=['duration', 'up_flow',
'down_flow'], aggfunc='mean').fillna(0)
features.columns = [f'{x[0]}_{x[1]}' for x in features.columns]

# Standardize data
scaler = StandardScaler()
features_scaled = scaler.fit_transform(features)

# Plot the histograms of each column before and after standardization
n_cols = 3 # Only plot the first 3 columns

fig, axs = plt.subplots(n_cols, 2, figsize=(10, 5 * n_cols))

for i in range(n_cols):
    # Plot the histogram of the i-th column before standardization
    axs[i, 0].hist(features.iloc[:, i], bins=30)
    axs[i, 0].set_title(f'Histogram of column {i} before standardization')

    # Plot the histogram of the i-th column after standardization
    axs[i, 1].hist(features_scaled[:, i], bins=30)
    axs[i, 1].set_title(f'Histogram of column {i} after standardization')

plt.show()

from sklearn.metrics import silhouette_score

# 肘部法则
distortions = []
for i in range(1, 11):
    kmeans = KMeans(n_clusters=i)
    kmeans.fit(features)
    distortions.append(kmeans.inertia )

plt.plot(range(1, 11), distortions, marker='o')
plt.xlabel('Number of clusters')
plt.ylabel('Distortion')
plt.show()

# 轮廓系数法
silhouette_scores = []
for i in range(2, 11):
    kmeans = KMeans(n_clusters=i)
    kmeans_labels = kmeans.fit_predict(features)
    silhouette_scores.append(silhouette_score(features, kmeans_labels))

plt.plot(range(2, 11), silhouette_scores, marker='o')
plt.xlabel('Number of clusters')
plt.ylabel('Silhouette Score')

```

```
plt.show()

from sklearn.manifold import TSNE
from sklearn.cluster import DBSCAN, AgglomerativeClustering
from sklearn.mixture import BayesianGaussianMixture
import matplotlib.pyplot as plt

# t-SNE 降维
tsne = TSNE(n_components=2)
features_tsne = tsne.fit_transform(features)

# K-Means 聚类
kmeans = KMeans(n_clusters=5)
kmeans_labels = kmeans.fit_predict(features)

plt.scatter(features_tsne[:, 0], features_tsne[:, 1], c=kmeans_labels)
plt.title('K-Means Clustering with t-SNE')
plt.show()

# t-SNE 降维
tsne = TSNE(n_components=2)
features_tsne = tsne.fit_transform(features)

# DBSCAN 聚类
dbscan = DBSCAN(eps=3, min_samples=3)
dbscan_labels = dbscan.fit_predict(features)

plt.scatter(features_tsne[:, 0], features_tsne[:, 1], c=dbscan_labels)
plt.title('DBSCAN Clustering with t-SNE')
plt.show()

# Agglomerative Clustering 聚类
agg = AgglomerativeClustering(n_clusters=5)
agg_labels = agg.fit_predict(features)

plt.scatter(features_tsne[:, 0], features_tsne[:, 1], c=agg_labels)
plt.title('Agglomerative Clustering with t-SNE')
plt.show()

# Gaussian Mixture 聚类
bgm = BayesianGaussianMixture(n_components=5, covariance_type='full')
bgm_labels = bgm.fit_predict(features)

plt.scatter(features_tsne[:, 0], features_tsne[:, 1], c=bgm_labels)
plt.title('Gaussian Mixture Clustering with t-SNE')
plt.show()

from sklearn.manifold import TSNE
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt

# t-SNE 降维
tsne = TSNE(n_components=3)
features_tsne = tsne.fit_transform(features)

# 创建图形
```

```

fig = plt.figure(figsize=(15, 10))

# 为四种聚类算法创建子图
for i, labels, title in zip(range(4), [kmeans_labels, dbscan_labels, agg_labels, bgm_labels],
                             ['KMeans', 'DBSCAN', 'Agglomerative Clustering', 'Gaussian Mixture']):
    ax = fig.add_subplot(2, 2, i+1, projection='3d')
    scatter = ax.scatter(features_tsne[:, 0], features_tsne[:, 1], features_tsne[:, 2], c=labels, cmap='viridis',
s=20, alpha=0.6)
    ax.set_title(f'{title} Clustering with t-SNE')
    fig.colorbar(scatter, ax=ax, format="%d")

plt.show()

from sklearn.metrics import silhouette_score, calinski_harabasz_score, davies_bouldin_score

scores = []
for labels in [kmeans_labels, dbscan_labels, agg_labels, bgm_labels]:
    silhouette = silhouette_score(features, labels)
    calinski_harabasz = calinski_harabasz_score(features, labels)
    davies_bouldin = davies_bouldin_score(features, labels)
    scores.append((silhouette, calinski_harabasz, davies_bouldin))

methods = ['KMeans', 'DBSCAN', 'AgglomerativeClustering', 'GaussianMixture']
metrics = ['Silhouette Score', 'Calinski-Harabasz Score', 'Davies-Bouldin Score']

for i in range(3):
    plt.figure(figsize=(9, 6))
    plt.bar(methods, [score[i] for score in scores], color=['cornflowerblue', 'lightcoral', 'lightgreen', 'gold'])
    plt.title(metrics[i], fontsize=16)
    plt.xlabel('Clustering Method', fontsize=14)
    plt.ylabel(metrics[i], fontsize=14)
    plt.xticks(fontsize=12)
    plt.yticks(fontsize=12)
    plt.grid(axis='y', linestyle='--', alpha=0.6)
    plt.show()

import vaex
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler

app_class = vaex.from_csv("C:\\Users\\[redacted]\\Desktop\\A\\app_class.csv", header=None, names=['appid',
'app_class'])

# Create empty DataFrame to store results
grouped_data = None

for i in range(1, 22):
    day = str(i).zfill(2)
    file_path = f'C:\\Users\\[redacted]\\Desktop\\A\\day{day}.txt'
    data = vaex.from_csv(file_path, header=None, names=['uid', 'appid', 'app_type', 'start_day', 'start_time',
'end day', 'end time', 'duration', 'up flow', 'down flow'])

```

```

data = data.join(app_class, on='appid', how='left', allow_duplication=True) # Added
allow_duplication=True

numeric_cols = ['duration', 'up_flow', 'down_flow']
for col in numeric_cols:
    data[col] = data[col].fillna(data[col].mean())

data = data[['uid', 'app_class', 'duration', 'up_flow', 'down_flow', 'app_type', 'start_time', 'end_time']]

# Compute mean duration, total up flow and down flow for each user and each app class
grouped = data.groupby(['uid', 'app_class']).agg({'duration': 'mean', 'up_flow': 'sum', 'down_flow':
'sum'})

if grouped is None:
    grouped_data = grouped
else:
    grouped_data = grouped_data.concat(grouped)

grouped_data = grouped_data.to_pandas_df()

features = grouped_data.pivot_table(index='uid', columns='app_class', values=['duration', 'up_flow',
'down_flow'], aggfunc='mean').fillna(0)
features.columns = [f'{x[0]}_{x[1]}' for x in features.columns]

scaler = StandardScaler()
features_scaled = scaler.fit_transform(features)

kmeans = KMeans(n_clusters=5)
kmeans_labels = kmeans.fit_predict(features_scaled)

cluster_centers = []
for i in range(kmeans.n_clusters):
    cluster_center = features_scaled[kmeans_labels == i].mean(axis=0)
    cluster_centers.append(cluster_center)

for i, cluster_center in enumerate(cluster_centers):
    print(f'Cluster {i}:')
    print(f'  平均持续时间: {cluster_center["duration_a"]:.2f}')
    print(f'  平均上行流量: {cluster_center["up_flow_a"]:.2f}')
    print(f'  平均下行流量: {cluster_center["down_flow_a"]:.2f}')
    print()

“=====分类=====”
“=====读取数据=====”

import pandas as pd
import numpy as np
from glob import glob

# 定义路径
app_class_path = "D:\程序\item\data_set/app_class.csv"

```

```

days_data_path = "D:\程序\item\data_set/day*.txt"

# 读取 app 类别数据
app_class = pd.read_csv(app_class_path, header=None, names=["appid", "app_class"], dtype={"appid": str,
"app_class": str})

# 删除 appid 列中包含非数字字符的行
app_class = app_class[app_class["appid"].str.isnumeric()]

# 数据类型转换以节省内存
dtypes = {
    "uid": str,
    "appid": str,
    "app_type": "category",
    "start_day": np.int8,
    "start_time": str,
    "end_day": np.int8,
    "end_time": str,
    "duration": np.int32,
    "up_flow": np.int32,
    "down_flow": np.int32,
}

data = pd.DataFrame() # 初始化一个空的 data DataFrame 来存储合并的数据

chunk_size = 500000 # 设置块大小
selected_uids = set() # 存储已选择的用户 ID

# 初始化选择 1 万个用户的标志
has_selected_users = False

for file in glob(days_data_path):
    for chunk in pd.read_csv(file, header=None, names=["uid", "appid", "app_type", "start_day",
"start_time",
"end_day", "end_time", "duration",
"up_flow", "down_flow"],
dtype=dtypes, chunksize=chunk_size):

        # 如果还没有选择 2 万个用户，则从当前块中随机选择
        if not has_selected_users:
            selected_uids = set(chunk["uid"].sample(n=20000, replace=False))
            has_selected_users = True

        # 只保留已选择的用户 ID 的数据
        chunk = chunk[chunk["uid"].isin(selected_uids)]

        # 删除 appid 列中包含非数字字符的行
        chunk = chunk[chunk["appid"].str.isnumeric()]

        # 与 app_class 进行合并
        chunk = pd.merge(chunk, app_class, on="appid", how="left")

        # 追加到 data DataFrame 中
        data = pd.concat([data, chunk], ignore_index=True)

# 释放内存

```

```

# 剔除 duration、up_flow 和 down_flow 为 0 的行
data = data[(data["duration"] > 0) & (data["up_flow"] > 0) & (data["down_flow"] > 0)]

# 剔除使用时长过于短（小于 5 秒）的行
data = data[data["duration"] >= 5]

# 对于未知类别的 APP，我们将其类别标记为"unknown"
data["app_class"] = data["app_class"].fillna("unknown")

# 将数据分为训练集和测试集
train_data = data[data["start_day"] <= 15]
test_data = data[data["start_day"] > 15]

# 提取每个用户在训练集和测试集中的特征
agg_funcs = {"duration": "sum", "up_flow": "sum", "down_flow": "sum"}
X_train = train_data.groupby(["uid", "app_class"]).agg(agg_funcs).reset_index().pivot_table(index="uid",
columns="app_class", values=["duration", "up_flow", "down_flow"], aggfunc="sum").fillna(0)
X_test = test_data.groupby(["uid", "app_class"]).agg(agg_funcs).reset_index().pivot_table(index="uid",
columns="app_class", values=["duration", "up_flow", "down_flow"], aggfunc="sum").fillna(0)

# 提取每个用户在训练集和测试集中的标签
y_train = train_data.groupby("uid")["app_class"].unique().apply(set)
y_test = test_data.groupby("uid")["app_class"].unique().apply(set)

# 为了适应后面的模型，我们将 y_train 和 y_test 转化为多标签的格式
all_classes = app_class["app_class"].unique()
y_train_multi = pd.DataFrame(index=y_train.index, columns=all_classes).fillna(0)
y_test_multi = pd.DataFrame(index=y_test.index, columns=all_classes).fillna(0)

for idx, classes in y_train.items():
    y_train_multi.loc[idx, list(classes)] = 1

for idx, classes in y_test.items():
    y_test_multi.loc[idx, list(classes)] = 1

“=====定义绘制模型学习函数=====”
import numpy as np
from sklearn.model_selection import learning_curve
import matplotlib.pyplot as plt
import seaborn as sns

def plot_learning_curve(estimator, title, X, y, ylim=None, cv=None, n_jobs=1, train_sizes=np.linspace(.1,
1.0, 5)):
    plt.figure(figsize=(12, 8)) # 增大图形尺寸
    plt.title(title, fontsize=20) # 增大标题字体并设置字体样式
    if ylim is not None:
        plt.ylim(*ylim)
    plt.xlabel("Training examples", fontsize=16) # 增大 X 轴标签字体并设置字体样式
    plt.ylabel("Score", fontsize=16) # 增大 Y 轴标签字体并设置字体样式
    train_sizes, train_scores, test_scores = learning_curve(estimator, X, y, cv=cv, n_jobs=n_jobs,
train_sizes=train_sizes)
    train_scores_mean = np.mean(train_scores, axis=1)
    train_scores_std = np.std(train_scores, axis=1)

```

```

test_scores_mean = np.mean(test_scores, axis=1)
test_scores_std = np.std(test_scores, axis=1)

plt.grid(color='gray', linestyle='--', linewidth=0.5) # 调整网格线颜色, 线型, 宽度

plt.fill_between(train_sizes, train_scores_mean - train_scores_std, train_scores_mean +
train_scores_std, alpha=0.1, color='r')
plt.fill_between(train_sizes, test_scores_mean - test_scores_std, test_scores_mean + test_scores_std,
alpha=0.1, color='g')
plt.plot(train_sizes, train_scores_mean, 'o-', color=sns.color_palette("husl", 8)[3], label="Training
score", linewidth=3) # 调整线宽
plt.plot(train_sizes, test_scores_mean, 'o-', color=sns.color_palette("husl", 8)[4],
label="Cross-validation score", linewidth=3) # 调整线宽

plt.legend(loc="best", fontsize=14) # 增大图例字体
plt.tight_layout() # 自动调整子图参数, 使之填充整个图像区域
plt.show()

```

“=====逻辑回归=====”

```

from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.metrics import accuracy_score, hamming_loss
from sklearn.multioutput import MultiOutputClassifier
import seaborn as sns
import numpy as np

# 确保 y_train_multi 和 y_test_multi 中不存在 NaN 值
y_train_multi = y_train_multi.fillna(0)
y_test_multi = y_test_multi.fillna(0)

# 定义逻辑回归模型并将其包装为多输出分类器
base_model = LogisticRegression(max_iter=10000, n_jobs=-1)
multi_model = MultiOutputClassifier(base_model)

# 设置参数网格
param_grid = {
    'estimator__C': [0.01, 0.1, 1, 10],
    'estimator__penalty': ['l1', 'l2']
}

# 使用 GridSearchCV 进行网格搜索
grid_search = GridSearchCV(multi_model, param_grid, cv=5, verbose=1, scoring="accuracy")

# 将训练数据进一步划分为训练集和验证集
X_train_new, X_val, y_train_new, y_val = train_test_split(X_train, y_train_multi, test_size=0.2,
random_state=42)

# 开始训练
grid_search.fit(X_train_new, y_train_new)

# 可视化超参数搜索过程
results = grid_search.cv_results_
C_values = list(param_grid['estimator__C'])
penalty_values = list(param_grid['estimator__penalty'])

```

```

scores_matrix = np.array(results['mean_test_score']).reshape(len(C_values), len(penalty_values))

plt.figure(figsize=(10, 6))
sns.heatmap(scores_matrix, annot=True, fmt=".3f", xticklabels=penalty_values, yticklabels=C_values,
            cmap="YlGnBu")
plt.title('Grid Search Cross Validation Scores')
plt.xlabel('Penalty')
plt.ylabel('C value')
plt.show()

# 画出学习曲线
plot_learning_curve(grid_search.best_estimator, "Learning Curve for Logistic Regression Model",
                    X_train_new, y_train_new, cv=5)

# 在验证集上预测
y_val_pred = grid_search.predict(X_val)
val_accuracy = accuracy_score(y_val.values, y_val_pred) # 使用总体准确性来评估模型性能
val_hamming_loss = hamming_loss(y_val.values, y_val_pred) # 使用 Hamming 损失来评估模型性能

# 在测试集上预测
y_test_pred = grid_search.predict(X_test)
test_accuracy = accuracy_score(y_test_multi.values, y_test_pred) # 使用总体准确性来评估模型性能
test_hamming_loss = hamming_loss(y_test_multi.values, y_test_pred) # 使用 Hamming 损失来评估模型性能

print("Validation set accuracy: ", val_accuracy)
print("Validation set Hamming loss: ", val_hamming_loss)
print("Test set accuracy: ", test_accuracy)
print("Test set Hamming loss: ", test_hamming_loss)
print("Best parameters from grid search: ", grid_search.best_params_)

“=====多项式 NB=====”
from sklearn.naive_bayes import MultinomialNB
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.metrics import accuracy_score, hamming_loss
from sklearn.multioutput import MultiOutputClassifier
from tqdm.notebook import tqdm, trange
import seaborn as sns
import matplotlib.pyplot as plt

# 确保 y_train_multi 和 y_test_multi 中不存在 NaN 值
y_train_multi = y_train_multi.fillna(0)
y_test_multi = y_test_multi.fillna(0)

# 定义 MultinomialNB 模型并将其包装为多输出分类器
base_model = MultinomialNB()
multi_model = MultiOutputClassifier(base_model)

# 设置参数网格
param_grid = {
    'estimator__alpha': [0.01, 0.1, 0.5, 1, 5, 10]
}

# 使用 GridSearchCV 进行网格搜索, 并设置 verbose 参数为 0, 以在 tqdm 中显示进度
grid_search = GridSearchCV(multi_model, param_grid, cv=5, verbose=0)

```

```

model = grid_search

# 在新的训练集上训练模型并显示进度
for _ in tqdm(range(model.cv * len(param_grid['estimator__alpha'])), desc="Training", unit="folds"):
    model.fit(X_train_new, y_train_new)

# 在验证集上预测
y_val_pred = model.predict(X_val)
val_accuracy = accuracy_score(y_val.values, y_val_pred)
val_hamming_loss = hamming_loss(y_val.values, y_val_pred)

# 画出学习曲线
plot_learning_curve(model, "Learning Curve for MultinomialNB Model", X_train_new, y_train_new, cv=5)

# 在测试集上预测
y_test_pred = model.predict(X_test)
test_accuracy = accuracy_score(y_test_multi.values, y_test_pred)
test_hamming_loss = hamming_loss(y_test_multi.values, y_test_pred)

# 绘制超参数对 score 的影响的折线图
results = model.cv_results_
params = results['params']
means = results['mean_test_score']
stds = results['std_test_score']

plt.figure(figsize=(10, 6))
for param, mean_score, std_score in zip(params, means, stds):
    plt.errorbar(str(param['estimator__alpha']), mean_score, yerr=std_score, fmt='o')
plt.xlabel('Alpha')
plt.ylabel('Mean Score')
plt.title('Alpha vs. Mean Score')
plt.show()

# 绘制交叉验证的结果的热力图
scores_matrix = means.reshape(len(param_grid['estimator__alpha']), 1)
sns.heatmap(scores_matrix, annot=True, fmt=".3f", xticklabels=['Mean Score'],
            yticklabels=param_grid['estimator__alpha'], cmap="Blues")
plt.xlabel('Mean Score')
plt.ylabel('Alpha')
plt.title('Grid Search CV Results Heatmap')
plt.show()

print("Validation set accuracy: ", val_accuracy)
print("Validation set Hamming loss: ", val_hamming_loss)
print("Test set accuracy: ", test_accuracy)
print("Test set Hamming loss: ", test_hamming_loss)
print("Best parameters from grid search: ", model.best_params_)

```

“=====随机森林=====”

```

from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.metrics import accuracy_score, hamming_loss
from sklearn.multioutput import MultiOutputClassifier
from tqdm.notebook import tqdm, trange
import seaborn as sns
import matplotlib.pyplot as plt

```

```

# 确保 y_train_multi 和 y_test_multi 中不存在 NaN 值
y_train_multi = y_train_multi.fillna(0)
y_test_multi = y_test_multi.fillna(0)

# 定义随机森林模型并将其包装为多输出分类器
base_model = RandomForestClassifier()
multi_model = MultiOutputClassifier(base_model)

# 设置参数网格
param_grid = {
    'estimator__n_estimators': [50, 100, 150],
    'estimator__max_depth': [5, 10, 15]
}

# 使用 GridSearchCV 进行网格搜索，并设置 verbose 参数为 0，以在 tqdm 中显示进度
grid_search = GridSearchCV(multi_model, param_grid, cv=5, verbose=0)
model = grid_search

# 在新的训练集上训练模型并显示进度
for _ in tqdm(range(model.cv * len(param_grid['estimator__n_estimators']) *
len(param_grid['estimator__max_depth'])), desc="Training", unit="folds"):
    model.fit(X_train_new, y_train_new)

# 在验证集上预测
y_val_pred = model.predict(X_val)
val_accuracy = accuracy_score(y_val.values, y_val_pred)
val_hamming_loss = hamming_loss(y_val.values, y_val_pred)

# 画出学习曲线
plot_learning_curve(model, "Learning Curve for RandomForest Model", X_train_new, y_train_new, cv=5)

# 在测试集上预测
y_test_pred = model.predict(X_test)
test_accuracy = accuracy_score(y_test_multi.values, y_test_pred)
test_hamming_loss = hamming_loss(y_test_multi.values, y_test_pred)

# 绘制超参数对 score 的影响
results = model.cv_results_
params = results['params']
means = results['mean_test_score']
stds = results['std_test_score']

# 绘制 n_estimators 的折线图
plt.figure(figsize=(10, 6))
for param, mean_score, std_score in zip(params, means, stds):
    if 'estimator__n_estimators' in param:
        plt.errorbar(param['estimator__n_estimators'], mean_score, yerr=std_score, fmt='o')
plt.xlabel('Number of Estimators')
plt.ylabel('Mean Score')
plt.title('Number of Estimators vs. Mean Score')
plt.show()

# 绘制热力图
scores_matrix = means.reshape(len(param_grid['estimator__max_depth']),
len(param_grid['estimator__n_estimators']))

```

```
sns.heatmap(scores_matrix, annot=True, fmt=".3f", xticklabels=param_grid['estimator__n_estimators'],
yticklabels=param_grid['estimator__max_depth'], cmap="Blues")
plt.xlabel('Number of Estimators')
plt.ylabel('Max Depth')
plt.title('Grid Search CV Results Heatmap')
plt.show()
```

```
print("Validation set accuracy: ", val_accuracy)
print("Validation set Hamming loss: ", val_hamming_loss)
print("Test set accuracy: ", test_accuracy)
print("Test set Hamming loss: ", test_hamming_loss)
print("Best parameters from grid search: ", model.best_params_)
```

“=====GradientBoosting=====”

```
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.metrics import accuracy_score, hamming_loss
from sklearn.multioutput import MultiOutputClassifier
from tqdm.notebook import tqdm, trange
import seaborn as sns
import matplotlib.pyplot as plt
```

```
# 确保 y_train_multi 和 y_test_multi 中不存在 NaN 值
y_train_multi = y_train_multi.fillna(0)
y_test_multi = y_test_multi.fillna(0)
```

```
# 定义 GradientBoostingClassifier 模型并将其包装为多输出分类器
base_model = GradientBoostingClassifier()
multi_model = MultiOutputClassifier(base_model)
```

```
# 设置参数网格
```

```
param_grid = {
    'estimator__n_estimators': [50, 100, 150],
    'estimator__learning_rate': [0.01, 0.1, 0.5],
}
```

```
# 使用 GridSearchCV 进行网格搜索，并设置 verbose 参数为 0，以在 tqdm 中显示进度
grid_search = GridSearchCV(multi_model, param_grid, cv=5, verbose=0, scoring='accuracy')
model = grid_search
```

```
# 在新的训练集上训练模型并显示进度
```

```
for _ in tqdm(range(model.cv * len(param_grid['estimator__n_estimators']) *
len(param_grid['estimator__learning_rate'])), desc="Training", unit="folds"):
    model.fit(X_train, y_train_multi)
```

```
# 在验证集上预测
```

```
y_val_pred = model.predict(X_test)
val_accuracy = accuracy_score(y_test_multi.values, y_val_pred)
val_hamming_loss = hamming_loss(y_test_multi.values, y_val_pred)
```

```
# 画出学习曲线
```

```
plot_learning_curve(model, "Learning Curve for GBDT Model", X_train, y_train_multi, cv=5)
```

```
# 在测试集上预测
```

```
y_test_pred = model.predict(X_test)
test_accuracy = accuracy_score(y_test_multi.values, y_test_pred)
```

```

test_hamming_loss = hamming_loss(y_test_multi.values, y_test_pred)

# 绘制超参数对 score 的影响的折线图
results = model.cv_results_
params = results['params']
means = results['mean_test_score']
stds = results['std_test_score']

plt.figure(figsize=(10, 6))
for param, mean_score, std_score in zip(params, means, stds):
    plt.errorbar(str(param['estimator__n_estimators']), mean_score, yerr=std_score, fmt='o')
plt.xlabel('n_estimators')
plt.ylabel('Mean Score')
plt.title('n_estimators vs. Mean Score')
plt.show()

# 绘制交叉验证的结果的热力图
scores_matrix = means.reshape(len(param_grid['estimator__n_estimators']),
                                len(param_grid['estimator__learning_rate']))
sns.heatmap(scores_matrix, annot=True, fmt=".3f", xticklabels=param_grid['estimator__learning_rate'],
            yticklabels=param_grid['estimator__n_estimators'], cmap="Blues")
plt.xlabel('Learning Rate')
plt.ylabel('n_estimators')
plt.title('Grid Search CV Results Heatmap')
plt.show()

print("Validation set accuracy: ", val_accuracy)
print("Validation set Hamming loss: ", val_hamming_loss)
print("Test set accuracy: ", test_accuracy)
print("Test set Hamming loss: ", test_hamming_loss)
print("Best parameters from grid search: ", model.best_params_)

“=====AdaBoost=====”
from sklearn.ensemble import AdaBoostClassifier
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.metrics import accuracy_score, hamming_loss
from sklearn.multioutput import MultiOutputClassifier
from tqdm.notebook import tqdm, trange
import seaborn as sns
import matplotlib.pyplot as plt

# 确保 y_train_multi 和 y_test_multi 中不存在 NaN 值
y_train_multi = y_train_multi.fillna(0)
y_test_multi = y_test_multi.fillna(0)

# 定义 AdaBoost 模型并将其包装为多输出分类器
base_model = AdaBoostClassifier()
multi_model = MultiOutputClassifier(base_model)

# 设置参数网格
param_grid = {
    'estimator__n_estimators': [50, 100, 150],
    'estimator__learning_rate': [0.01, 0.05, 0.1, 0.5, 1]
}

# 使用 GridSearchCV 进行网格搜索，并设置 verbose 参数为 0，以在 tqdm 中显示进度

```

```

grid_search = GridSearchCV(multi_model, param_grid, cv=5, verbose=0)
model = grid_search

# 在新的训练集上训练模型并显示进度
for _ in tqdm(range(model.cv * len(param_grid['estimator__n_estimators']) *
len(param_grid['estimator__learning_rate'])), desc="Training", unit="folds"):
    model.fit(X_train_new, y_train_new)

# 在验证集上预测
y_val_pred = model.predict(X_val)
val_accuracy = accuracy_score(y_val.values, y_val_pred) # 使用总体准确性来评估模型性能
val_hamming_loss = hamming_loss(y_val.values, y_val_pred) # 使用 Hamming 损失来评估模型性能

# 画出学习曲线
plot_learning_curve(model, "Learning Curve for AdaBoost Model", X_train_new, y_train_new, cv=5)

# 在测试集上预测
y_test_pred = model.predict(X_test)
test_accuracy = accuracy_score(y_test_multi.values, y_test_pred) # 使用总体准确性来评估模型性能
test_hamming_loss = hamming_loss(y_test_multi.values, y_test_pred) # 使用 Hamming 损失来评估模型性能

# 绘制超参数对 score 的影响
results = model.cv_results_
params = results['params']
means = results['mean_test_score']
stds = results['std_test_score']

for param, mean_score, std_score in zip(params, means, stds):
    plt.errorbar(str(param), mean_score, yerr=std_score, fmt='o')

plt.xlabel('Parameters')
plt.ylabel('Mean Score')
plt.title('Parameters vs. Mean Score')
plt.show()

# 绘制交叉验证的结果的热力图
scores_matrix = means.reshape(len(param_grid['estimator__learning_rate']),
len(param_grid['estimator__n_estimators']))
sns.heatmap(scores_matrix, annot=True, fmt=".3f", xticklabels=param_grid['estimator__n_estimators'],
yticklabels=param_grid['estimator__learning_rate'], cmap="Blues")
plt.xlabel('Number of Estimators')
plt.ylabel('Learning Rate')
plt.title('Grid Search CV Results Heatmap')
plt.show()

print("Validation set accuracy: ", val_accuracy)
print("Validation set Hamming loss: ", val_hamming_loss)
print("Test set accuracy: ", test_accuracy)
print("Test set Hamming loss: ", test_hamming_loss)
print("Best parameters from grid search: ", model.best_params_)

“=====XGBoost=====”
import xgboost as xgb
from xgboost import XGBClassifier

```

```

from sklearn.multioutput import MultiOutputClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score, hamming_loss
from tqdm.notebook import tqdm, trange
import seaborn as sns
import matplotlib.pyplot as plt

# 确保 y_train_multi 和 y_test_multi 中不存在 NaN 值
y_train_multi = y_train_multi.fillna(0)
y_test_multi = y_test_multi.fillna(0)

# 使用 XGBClassifier, 并开启 GPU 加速
base_model = XGBClassifier(tree_method='gpu_hist', predictor='gpu_predictor')
multi_model = MultiOutputClassifier(base_model)

# 设置参数网格
param_grid = {
    'estimator__learning_rate': [0.01, 0.1, 0.5],
    'estimator__max_depth': [3, 6, 9],
    'estimator__n_estimators': [50, 100]
}

grid_search = GridSearchCV(multi_model, param_grid, cv=5, verbose=0, scoring='accuracy')

# 使用 tqdm.notebook 来展示训练过程
total_iterations = len(param_grid['estimator__learning_rate']) * len(param_grid['estimator__max_depth']) * len(param_grid['estimator__n_estimators']) * grid_search.cv
for _ in tqdm(range(total_iterations), desc="Training", unit="folds"):
    grid_search.fit(X_train, y_train_multi)

# 预测
y_val_pred = grid_search.predict(X_test)
val_accuracy = accuracy_score(y_test_multi.values, y_val_pred)
val_hamming_loss = hamming_loss(y_test_multi.values, y_val_pred)

# 画出学习曲线
plot_learning_curve(grid_search, "Learning Curve for XGBoost Model", X_train, y_train_multi, cv=5)

# 绘制超参数对 score 的影响的折线图
results = grid_search.cv_results_
params = results['params']
means = results['mean_test_score']
stds = results['std_test_score']

plt.figure(figsize=(10, 6))
for param, mean_score, std_score in zip(params, means, stds):
    plt.errorbar(str(param['estimator__learning_rate']), mean_score, yerr=std_score, fmt='o')
plt.xlabel('Learning Rate')
plt.ylabel('Mean Score')
plt.title('Learning Rate vs. Mean Score')
plt.show()

# 绘制交叉验证的结果的热力图
scores_matrix = means.reshape(len(param_grid['estimator__learning_rate']),
                               len(param_grid['estimator__max_depth']))
sns.heatmap(scores_matrix, annot=True, fmt=".3f", xticklabels=param_grid['estimator__learning_rate'],

```

```

yticklabels=param_grid['estimator__max_depth'], cmap="Blues")
plt.xlabel('Learning Rate')
plt.ylabel('Max Depth')
plt.title('Grid Search CV Results Heatmap')
plt.show()

print("Validation set accuracy: ", val_accuracy)
print("Validation set Hamming loss: ", val_hamming_loss)
print("Best parameters from grid search: ", grid_search.best_params_)

```

“=====BP 神经网络=====”

```

import torch
import torch.nn as nn
import torch.optim as optim
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.metrics import accuracy_score, hamming_loss
from sklearn.multioutput import MultiOutputClassifier
from tqdm.notebook import tqdm, trange
import seaborn as sns
import matplotlib.pyplot as plt

```

定义 BP 神经网络模型

```

class BPNN(nn.Module):
    def __init__(self, input_dim, hidden_dim, output_dim):
        super(BPNN, self).__init__()
        self.fc1 = nn.Linear(input_dim, hidden_dim)
        self.fc2 = nn.Linear(hidden_dim, output_dim)
        self.sigmoid = nn.Sigmoid()

    def forward(self, x):
        x = self.fc1(x)
        x = self.sigmoid(x)
        x = self.fc2(x)
        return x

```

定义 PyTorch 模型的包装器，使其与 scikit-learn 兼容

```

class PyTorchClassifier:
    def __init__(self, model, criterion, optimizer, epochs=10, device='cuda'):
        self.model = model.to(device)
        self.criterion = criterion
        self.optimizer = optimizer
        self.epochs = epochs
        self.device = device

    def get_params(self, deep=True):
        return {"model": self.model, "criterion": self.criterion, "optimizer": self.optimizer, "epochs":
self.epochs, "device": self.device}

    def set_params(self, **parameters):
        for parameter, value in parameters.items():
            setattr(self, parameter, value)
        return self

    def fit(self, X, y):
        X_tensor = torch.tensor(X.values, dtype=torch.float32).to(self.device)
        y_tensor = torch.tensor(y.values, dtype=torch.float32).to(self.device)

```

```

        for epoch in trange(self.epochs, desc="Training"):
            self.optimizer.zero_grad()
            outputs = self.model(X_tensor)
            loss = self.criterion(outputs, y_tensor)
            loss.backward()
            self.optimizer.step()
        return self

    def predict(self, X):
        X_tensor = torch.tensor(X.values, dtype=torch.float32).to(self.device)
        with torch.no_grad():
            outputs = self.model(X_tensor)
        return (outputs > 0.5).int().cpu().numpy()

    def score(self, X, y):
        y_pred = self.predict(X)
        return accuracy_score(y, y_pred)

# 设置超参数范围
param_grid = {
    'hidden_dim': [32, 64, 128],
    'lr': [0.001, 0.01, 0.1]
}

best_score = 0
best_params = None
results = []

# 使用网格搜索
for hidden_dim in param_grid['hidden_dim']:
    for lr in param_grid['lr']:
        model = BPNN(X_train.shape[1], hidden_dim, y_train_multi.shape[1])
        criterion = nn.BCEWithLogitsLoss()
        optimizer = optim.Adam(model.parameters(), lr=lr)
        classifier = PyTorchClassifier(model, criterion, optimizer, epochs=10)
        classifier.fit(X_train, y_train_multi)
        y_val_pred = classifier.predict(X_val)
        score = accuracy_score(y_val.values, y_val_pred)
        results.append({
            'hidden_dim': hidden_dim,
            'lr': lr,
            'accuracy': score
        })
        if score > best_score:
            best_score = score
            best_params = {'hidden_dim': hidden_dim, 'lr': lr}

# 绘制折线图
results_df = pd.DataFrame(results)
plt.figure(figsize=(12, 6))
for hidden_dim in param_grid['hidden_dim']:
    subset = results_df[results_df['hidden_dim'] == hidden_dim]
    plt.plot(subset['lr'], subset['accuracy'], '-o', label=f'Hidden Dim: {hidden_dim}')
plt.xlabel('Learning Rate')
plt.ylabel('Accuracy')
plt.legend()
plt.title('Accuracy vs. Learning Rate for different Hidden Dimensions')

```

```

plt.show()

# 绘制热力图
heatmap_data = results_df.pivot(index="hidden_dim", columns="lr", values="accuracy")
sns.heatmap(heatmap_data, annot=True, cmap="Blues")
plt.title('Grid Search CV Results Heatmap')
plt.show()

# 画出学习曲线
best_model = BPNN(X_train.shape[1], best_params['hidden_dim'], y_train_multi.shape[1])
best_criterion = nn.BCEWithLogitsLoss()
best_optimizer = optim.Adam(best_model.parameters(), lr=best_params['lr'])
best_classifier = PyTorchClassifier(best_model, best_criterion, best_optimizer, epochs=10)
plot_learning_curve(best_classifier, "Learning Curve for BPNN Model", X_train, y_train_multi, cv=5)

# 在测试集上预测
y_test_pred = best_classifier.predict(X_test)
test_accuracy = accuracy_score(y_test_multi.values, y_test_pred)
test_hamming_loss = hamming_loss(y_test_multi.values, y_test_pred)

print("Validation set accuracy: ", best_score)
print("Test set accuracy: ", test_accuracy)
print("Test set Hamming loss: ", test_hamming_loss)
print("Best parameters from grid search: ", best_params)

```

“=====CNN=====”

```

import torch
from torch import nn, optim
from torch.utils.data import TensorDataset, DataLoader
from tqdm.notebook import tqdm
import numpy as np
import matplotlib.pyplot as plt

```

```

# 定义神经网络模型
class BPNN(nn.Module):
    def __init__(self, input_size, output_size):
        super(BPNN, self).__init__()
        self.fc = nn.Sequential(
            nn.Linear(input_size, 256),
            nn.ReLU(),
            nn.Dropout(0.5),
            nn.Linear(256, 128),
            nn.ReLU(),
            nn.Dropout(0.5),
            nn.Linear(128, 64),
            nn.ReLU(),
            nn.Dropout(0.5),
            nn.Linear(64, 32),
            nn.ReLU(),
            nn.Dropout(0.5),
            nn.Linear(32, output_size)
        )

    def forward(self, x):
        return self.fc(x)

```

```

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

model = BPNN(X_train.shape[1], 21).to(device) # 21 个输出类别

# 定义损失函数和优化器
criterion = nn.BCEWithLogitsLoss()
optimizer = optim.Adam(model.parameters(), lr=0.01)

# 转换数据到 torch.Tensor 并移动到相应设备上
X_train_tensor = torch.from_numpy(X_train.values).float().to(device)
y_train_tensor = torch.from_numpy(y_train_multi.values).float().to(device)
X_test_tensor = torch.from_numpy(X_test.values).float().to(device)
y_test_tensor = torch.from_numpy(y_test_multi.values).float().to(device)

# 创建数据加载器
train_data = TensorDataset(X_train_tensor, y_train_tensor)
train_loader = DataLoader(train_data, batch_size=32, shuffle=True)

# 训练模型
model.train()
train_losses = []
test_losses = []

# 使用 tqdm 显示进度条
for epoch in tqdm(range(100), desc="Epochs"): # 修改迭代次数为 100 次
    running_loss = 0
    for i, data in enumerate(tqdm(train_loader, desc=f"Epoch {epoch + 1} Training", leave=False)):
        inputs, labels = data
        optimizer.zero_grad()

        outputs = model(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        running_loss += loss.item()

    else:
        test_loss = 0
        accuracy = 0

        # Turn off gradients for validation, saves memory and computations
        with torch.no_grad():
            model.eval()
            predictions = model(X_test_tensor)
            test_loss += criterion(predictions, y_test_tensor)

        # 使用 sigmoid 函数将输出转换为 0-1 之间，然后将其与 0.5 进行比较以得到最终的预
        predicted_labels = torch.sigmoid(predictions) > 0.5

        accuracy = (predicted_labels == y_test_tensor).float().mean()

    model.train()

```

```

train_losses.append(running_loss / len(train_loader))
test_losses.append(test_loss.item())

print("\nEpoch: {} / 100.. ".format(epoch + 1),
      "Training Loss: {:.3f}.. ".format(running_loss / len(train_loader)),
      "Test Loss: {:.3f}.. ".format(test_loss),
      "Test Accuracy: {:.3f}".format(accuracy))

# 绘制训练过程中的损失曲线
plt.figure(figsize=(10, 7))
plt.grid(True, which="both", ls="--", c='0.65') # 增加背景网格
plt.plot(train_losses, label='Training loss')
plt.plot(test_losses, label='Validation loss')
plt.legend(frameon=False)
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('Training and Validation Loss over Epochs')
plt.show()

“=====CNN=====”
import torch
from torch import nn, optim
from torch.utils.data import TensorDataset, DataLoader
from tqdm.notebook import tqdm # 使用 tqdm 的 notebook 版本

# 定义 CNN 模型
class CNN(nn.Module):
    def __init__(self, input_size, num_classes):
        super(CNN, self).__init__()
        self.conv1 = nn.Sequential(
            nn.Conv1d(1, 32, kernel_size=3, stride=1, padding=1),
            nn.ReLU(),
            nn.MaxPool1d(kernel_size=2)
        )
        self.conv2 = nn.Sequential(
            nn.Conv1d(32, 64, kernel_size=3, stride=1, padding=1),
            nn.ReLU(),
            nn.MaxPool1d(kernel_size=2)
        )
        self.conv3 = nn.Sequential(
            nn.Conv1d(64, 128, kernel_size=3, stride=1, padding=1),
            nn.ReLU(),
            nn.MaxPool1d(kernel_size=2)
        )
        self.fc1 = nn.Linear(int(input_size / 8) * 128, 512)
        self.fc2 = nn.Linear(512, num_classes)

    def forward(self, x):
        x = x.unsqueeze(1)
        out = self.conv1(x)
        out = self.conv2(out)
        out = self.conv3(out)
        out = out.view(out.size(0), -1)
        out = self.fc1(out)

```

```

        out = self.fc2(out)
    return out

# 将数据转换为 PyTorch Tensor
X_train_tensor = torch.FloatTensor(X_train.values).to(device)
y_train_tensor = torch.FloatTensor(y_train_multi.values).to(device)
X_test_tensor = torch.FloatTensor(X_test.values).to(device)
y_test_tensor = torch.FloatTensor(y_test_multi.values).to(device)

# 使用 DataLoader 封装数据
batch_size = 32
train_dataset = TensorDataset(X_train_tensor, y_train_tensor)
train_loader = DataLoader(train_dataset, shuffle=True, batch_size=batch_size)

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

num_classes = 21 # 20 个 APP 类别和 1 个 unknown 类别
model = CNN(X_train.shape[1], num_classes).to(device)

# 定义损失函数和优化器
criterion = nn.BCEWithLogitsLoss() # 使用 BCEWithLogitsLoss 作为损失函数
optimizer = optim.Adam(model.parameters(), lr=0.01)

# 训练模型
model.train()
train_losses = []
test_losses = []

# 使用 tqdm 来显示进度
for epoch in tqdm(range(100), desc="Epochs"):
    running_loss = 0
    for i, data in enumerate(train_loader, 0):
        inputs, labels = data
        optimizer.zero_grad()

        outputs = model(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

    running_loss += loss.item()
else:
    test_loss = 0
    accuracy = 0

    # Turn off gradients for validation, saves memory and computations
    with torch.no_grad():
        model.eval()
        predictions = model(X_test_tensor)
        test_loss += criterion(predictions, y_test_tensor)

    accuracy = ((predictions > 0.5) == y_test_tensor.byte()).float().mean()

model.train()

```

```

train_losses.append(running_loss / len(train_loader))
test_losses.append(test_loss.item())

print("Epoch: {}/{}.. ".format(epoch + 1, 100),
      "Training Loss: {:.3f}.. ".format(running_loss / len(train_loader)),
      "Test Loss: {:.3f}.. ".format(test_loss),
      "Test Accuracy: {:.3f}".format(accuracy))

# 绘制训练过程中的损失曲线
plt.figure(figsize=(10, 7))
plt.plot(train_losses, label='Training loss', color='blue')
plt.plot(test_losses, label='Validation loss', color='red')
plt.legend(frameon=True, fontsize=14)
plt.grid(True, which="both", linestyle='--', linewidth=0.5)
plt.xlabel('Epochs', fontsize=14)
plt.ylabel('Loss', fontsize=14)
plt.title('Training and Validation Loss over Epochs', fontsize=16)
plt.show()

```

```

“=====GRU=====”
”

import torch
from torch import nn
from torch.optim import Adam
from torch.utils.data import DataLoader, TensorDataset, random_split
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
from tqdm.notebook import tqdm

# 数据转为 Tensor
X_train_tensor = torch.FloatTensor(X_train.values)
y_train_tensor = torch.FloatTensor(y_train_multi.values)
X_test_tensor = torch.FloatTensor(X_test.values)
y_test_tensor = torch.FloatTensor(y_test_multi.values)

# 将原始训练数据分为训练集和验证集
train_size = int(0.8 * len(X_train_tensor))
val_size = len(X_train_tensor) - train_size
train_dataset, val_dataset = random_split(TensorDataset(X_train_tensor, y_train_tensor), [train_size,
val_size])

test_dataset = TensorDataset(X_test_tensor, y_test_tensor)
train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True)
val_loader = DataLoader(val_dataset, batch_size=32)
test_loader = DataLoader(test_dataset, batch_size=32)

# 设备设置
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

class LSTMModel(nn.Module):
    def __init__(self, input_dim, hidden_dim, layer_dim, output_dim):
        super(LSTMModel, self).__init__()
        self.hidden_dim = hidden_dim
        self.layer_dim = layer_dim
        self.lstm = nn.LSTM(input_dim, hidden_dim, layer_dim, batch_first=True)

```

```

        self.fc = nn.Linear(hidden_dim, output_dim)

    def forward(self, x):
        h0 = torch.zeros(self.layer_dim, x.size(0), self.hidden_dim).to(device)
        c0 = torch.zeros(self.layer_dim, x.size(0), self.hidden_dim).to(device)
        out, (hn, cn) = self.lstm(x, (h0, c0))
        out = torch.sigmoid(self.fc(out[:, -1, :]))
        return out

input_dim = X_train_tensor.shape[1]
hidden_dim = 100 # 增加隐藏维度
layer_dim = 3 # 增加层数
output_dim = 21 # 有 21 个类别

model = LSTMMModel(input_dim, hidden_dim, layer_dim, output_dim).to(device)
criterion = nn.BCEWithLogitsLoss()
optimizer = Adam(model.parameters(), lr=0.001)

num_epochs = 100 # 增大迭代次数
train_loss = []
val_loss = []
accuracy_list = []

for epoch in range(num_epochs):
    model.train()
    epoch_train_loss = 0
    for i, (features, labels) in enumerate(tqdm(train_loader, desc=f"Epoch {epoch + 1}")):
        features = features.unsqueeze(1).to(device)
        labels = labels.to(device)
        optimizer.zero_grad()
        outputs = model(features)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
        epoch_train_loss += loss.item()
    train_loss.append(epoch_train_loss / len(train_loader))

    model.eval()
    epoch_val_loss = 0
    with torch.no_grad():
        for features, labels in val_loader:
            features = features.unsqueeze(1).to(device)
            labels = labels.to(device)
            outputs = model(features)
            loss = criterion(outputs, labels)
            epoch_val_loss += loss.item()
    val_loss.append(epoch_val_loss / len(val_loader))

y_pred = []
y_true = []
with torch.no_grad():
    for features, labels in test_loader:
        features = features.unsqueeze(1).to(device)
        labels = labels.to(device)
        outputs = model(features)
        predicted = (outputs > 0.5).int()

```

```

y_pred.extend(predicted.cpu().numpy())
y_true.extend(labels.cpu().numpy())

accuracy = accuracy_score(np.array(y_true).flatten(), np.array(y_pred).flatten())
accuracy_list.append(accuracy)
print(
    f'Epoch: {epoch + 1}, Train Loss: {train_loss[-1]:.4f}, Val Loss: {val_loss[-1]:.4f}, Accuracy: {accuracy:.4f}')

# 绘制损失和精度的变化
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.plot(train_loss, label="Train Loss")
plt.plot(val_loss, label="Validation Loss")
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.legend()
plt.grid(True, linestyle='--', alpha=0.7)
plt.subplot(1, 2, 2)
plt.plot(accuracy_list, label="Test Accuracy")
plt.xlabel("Epoch")
plt.ylabel("Accuracy")
plt.grid(True, linestyle='--', alpha=0.7)
plt.legend()
plt.show()

```

“=====Transformer=====”

```

import torch
import torch.nn as nn
import torch.optim as optim
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.metrics import accuracy_score, hamming_loss
from sklearn.multioutput import MultiOutputClassifier
from tqdm.notebook import tqdm, trange
import seaborn as sns
import matplotlib.pyplot as plt

# 定义 BP 神经网络模型
class BPNN(nn.Module):
    def __init__(self, input_dim, hidden_dim, output_dim):
        super(BPNN, self).__init__()
        self.fc1 = nn.Linear(input_dim, hidden_dim)
        self.fc2 = nn.Linear(hidden_dim, output_dim)
        self.sigmoid = nn.Sigmoid()

    def forward(self, x):
        x = self.fc1(x)
        x = self.sigmoid(x)
        x = self.fc2(x)
        return x

# 定义 PyTorch 模型的包装器，使其与 scikit-learn 兼容
class PyTorchClassifier:
    def __init__(self, model, criterion, optimizer, epochs=10, device='cuda'):
        self.model = model.to(device)
        self.criterion = criterion

```

```

        self.optimizer = optimizer
        self.epochs = epochs
        self.device = device

    def get_params(self, deep=True):
        return {"model": self.model, "criterion": self.criterion, "optimizer": self.optimizer, "epochs":
self.epochs, "device": self.device}

    def set_params(self, **parameters):
        for parameter, value in parameters.items():
            setattr(self, parameter, value)
        return self

    def fit(self, X, y):
        X_tensor = torch.tensor(X.values, dtype=torch.float32).to(self.device)
        y_tensor = torch.tensor(y.values, dtype=torch.float32).to(self.device)
        for epoch in range(self.epochs, desc="Training"):
            self.optimizer.zero_grad()
            outputs = self.model(X_tensor)
            loss = self.criterion(outputs, y_tensor)
            loss.backward()
            self.optimizer.step()
        return self

    def predict(self, X):
        X_tensor = torch.tensor(X.values, dtype=torch.float32).to(self.device)
        with torch.no_grad():
            outputs = self.model(X_tensor)
        return (outputs > 0.5).int().cpu().numpy()

    def score(self, X, y):
        y_pred = self.predict(X)
        return accuracy_score(y, y_pred)

# 设置超参数范围
param_grid = {
    'hidden_dim': [32, 64, 128],
    'lr': [0.001, 0.01, 0.1]
}

best_score = 0
best_params = None
results = []

# 使用网格搜索
results = [] # 用于存储不同超参数组合的结果

for hidden_dim in tqdm(param_grid['hidden_dim'], desc="Hidden Dim"): # 使用 tqdm 展示超参数搜索
    进度
        for lr in tqdm(param_grid['lr'], desc="Learning Rate", leave=False): # leave 参数设置为 False, 保持
            进度条不断行
                model = BPNN(X_train.shape[1], hidden_dim, y_train_multi.shape[1])
                criterion = nn.BCEWithLogitsLoss()
                optimizer = optim.Adam(model.parameters(), lr=lr)
                classifier = PyTorchClassifier(model, criterion, optimizer, epochs=10)
                classifier.fit(X_train, y_train_multi)
                y_val_pred = classifier.predict(X_val)

```

```

        score = accuracy_score(y_val.values, y_val_pred)
        results.append({
            'hidden_dim': hidden_dim,
            'lr': lr,
            'accuracy': score
        })

# 绘制折线图
results_df = pd.DataFrame(results)
plt.figure(figsize=(12, 6))
for hidden_dim in param_grid['hidden_dim']:
    subset = results_df[results_df['hidden_dim'] == hidden_dim]
    plt.plot(subset['lr'], subset['accuracy'], '-o', label=f'Hidden Dim: {hidden_dim}')
plt.xlabel('Learning Rate')
plt.ylabel('Accuracy')
plt.legend()
plt.title('Accuracy vs. Learning Rate for different Hidden Dimensions')
plt.show()

# 绘制热力图
heatmap_data = results_df.pivot(index="hidden_dim", columns="lr", values="accuracy")
sns.heatmap(heatmap_data, annot=True, cmap="Blues")
plt.title('Grid Search CV Results Heatmap')
plt.show()

# 画出学习曲线
best_model = BPNN(X_train.shape[1], best_params['hidden_dim'], y_train_multi.shape[1])
best_criterion = nn.BCEWithLogitsLoss()
best_optimizer = optim.Adam(best_model.parameters(), lr=best_params['lr'])
best_classifier = PyTorchClassifier(best_model, best_criterion, best_optimizer, epochs=10)
plot_learning_curve(best_classifier, "Learning Curve for BPNN Model", X_train, y_train_multi, cv=5)

# 在测试集上预测
y_test_pred = best_classifier.predict(X_test)
test_accuracy = accuracy_score(y_test_multi.values, y_test_pred)
test_hamming_loss = hamming_loss(y_test_multi.values, y_test_pred)

print("Validation set accuracy: ", best_score)
print("Test set accuracy: ", test_accuracy)
print("Test set Hamming loss: ", test_hamming_loss)
print("Best parameters from grid search: ", best_params)

“=====回归=====”
“=====读取数据及其预处理=====”

import pandas as pd
import numpy as np
from glob import glob

# 定义路径
app_class_path = "C:/Users/Desktop/A/app_class.csv"
days_data_path = "C:/Users/Desktop/A/day*.txt"

# 读取 app 类别数据
app_class = pd.read_csv(app_class_path, header=None, names=["appid", "app_class"], dtype={"appid": str,
"app_class": str})

```

```

# 删除 appid 列中包含非数字字符的行
app_class = app_class[app_class["appid"].str.isnumeric()]

# 数据类型转换以节省内存
dtypes = {
    "uid": str,
    "appid": str,
    "app_type": "category",
    "start_day": np.int8,
    "start_time": str,
    "end_day": np.int8,
    "end_time": str,
    "duration": np.int32,
    "up_flow": np.int32,
    "down_flow": np.int32,
}

data = pd.DataFrame() # 初始化一个空的 data DataFrame 来存储合并的数据

chunk_size = 500000 # 设置块大小
selected_uids = set() # 存储已选择的用户 ID

# 初始化选择 1 万个用户的标志
has_selected_users = False

for file in glob(days_data_path):
    for chunk in pd.read_csv(file, header=None, names=["uid", "appid", "app_type", "start_day",
"start time",
"end day", "end time", "duration",
"up_flow", "down_flow"],
dtype=dtypes, chunksize=chunk_size):
        # 如果还没有选择 2 万个用户，则从当前块中随机选择
        if not has_selected_users:
            selected_uids = set(chunk["uid"].sample(n=20000, replace=False))
            has_selected_users = True

        # 只保留已选择的用户 ID 的数据
        chunk = chunk[chunk["uid"].isin(selected_uids)]

        # 删除 appid 列中包含非数字字符的行
        chunk = chunk[chunk["appid"].str.isnumeric()]

        # 与 app_class 进行合并
        chunk = pd.merge(chunk, app_class, on="appid", how="left")

        # 追加到 data DataFrame 中
        data = pd.concat([data, chunk], ignore_index=True)

        # 释放内存
        del chunk

# 剔除 duration、up_flow 和 down_flow 为 0 的行
data = data[(data["duration"] > 0) & (data["up_flow"] > 0) & (data["down_flow"] > 0)]

# 剔除使用时长过于短（小于 5 秒）的行

```

```

data = data[data["duration"] >= 5]

# 对于未知类别的 APP，我们将其类别标记为"unknown"
data["app_class"] = data["app_class"].fillna("unknown")

# 将数据分为训练集和测试集
train_data = data[data["start_day"] <= 15]
test_data = data[data["start_day"] > 15]

# 提取每个用户在训练集和测试集中的特征
agg_funcs = {"duration": "sum"}
X_train = train_data.groupby(["uid", "app_class"]).agg(agg_funcs).reset_index().pivot_table(index="uid",
columns="app_class", values="duration", aggfunc="sum").fillna(0)
X_test = test_data.groupby(["uid", "app_class"]).agg(agg_funcs).reset_index().pivot_table(index="uid",
columns="app_class", values="duration", aggfunc="sum").fillna(0)

# 计算训练集和测试集中的日均使用时长
X_train /= 15 # 因为训练集有 15 天
X_test /= 15 # 因为测试集有 15 天

# 目标变量是测试集中的日均使用时长
y_train = X_train.copy()
y_test = X_test.copy()

```

“=====学习曲线函数=====”

```

import numpy as np
from sklearn.model_selection import learning_curve
import matplotlib.pyplot as plt
import seaborn as sns

def plot_learning_curve(estimator, title, X, y, ylim=None, cv=None, n_jobs=1, train_sizes=np.linspace(.1,
1.0, 5)):
    plt.figure(figsize=(12, 8)) # 增大图形尺寸
    plt.title(title, fontsize=20) # 增大标题字体并设置字体样式
    if ylim is not None:
        plt.ylim(*ylim)
    plt.xlabel("Training examples", fontsize=16) # 增大 X 轴标签字体并设置字体样式
    plt.ylabel("Score", fontsize=16) # 增大 Y 轴标签字体并设置字体样式
    train_sizes, train_scores, test_scores = learning_curve(estimator, X, y, cv=cv, n_jobs=n_jobs,
train_sizes=train_sizes)
    train_scores_mean = np.mean(train_scores, axis=1)
    train_scores_std = np.std(train_scores, axis=1)
    test_scores_mean = np.mean(test_scores, axis=1)
    test_scores_std = np.std(test_scores, axis=1)

    plt.grid(color='gray', linestyle='--', linewidth=0.5) # 调整网格线颜色，线型，宽度

    plt.fill_between(train_sizes, train_scores_mean - train_scores_std, train_scores_mean +
train_scores_std, alpha=0.1, color="r")
    plt.fill_between(train_sizes, test_scores_mean - test_scores_std, test_scores_mean + test_scores_std,
alpha=0.1, color="g")
    plt.plot(train_sizes, train_scores_mean, 'o-', color=sns.color_palette("husl", 8)[3], label="Training
score", linewidth=3) # 调整线宽
    plt.plot(train_sizes, test_scores_mean, 'o-', color=sns.color_palette("husl", 8)[4],
label="Cross-validation score", linewidth=3) # 调整线宽

```

```
plt.legend(loc="best", fontsize=14) # 增大图例字体
plt.tight_layout() # 自动调整子图参数，使之填充整个图像区域
plt.show()
```

“=====多元回归=====”

```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
from tqdm.notebook import tqdm
from sklearn.model_selection import KFold

# 使用 KFold 进行交叉验证，并在每次训练过程中使用 tqdm 显示进度
kf = KFold(n_splits=5, shuffle=True, random_state=42)

# 初始化模型
model = LinearRegression()

mse_scores = []
train_mse_scores = []

# 使用 tqdm 展示训练进度
for train_idx, val_idx in tqdm(kf.split(X_train), total=5, desc="Training model"):
    X_train_fold, X_val_fold = X_train.iloc[train_idx], X_train.iloc[val_idx]
    y_train_fold, y_val_fold = y_train.iloc[train_idx], y_train.iloc[val_idx]

    model.fit(X_train_fold, y_train_fold)

    preds_val = model.predict(X_val_fold)
    mse_val = mean_squared_error(y_val_fold, preds_val, multioutput='raw_values')
    mse_scores.append(mse_val)

    preds_train = model.predict(X_train_fold)
    mse_train = mean_squared_error(y_train_fold, preds_train, multioutput='raw_values')
    train_mse_scores.append(mse_train)

# 使用训练的模型绘制学习曲线
plot_learning_curve(model, "Learning Curve (Linear Regression)", X_train, y_train, cv=kf, n_jobs=-1)

# 计算平均的 NMSE
avg_mse = np.mean(mse_scores, axis=0)
y_mean = np.mean(y_train)
avg_nmse = np.sqrt(avg_mse) / np.sqrt(np.mean((y_train - y_mean.values)**2, axis=0))
avg_nmse_overall = np.mean(avg_nmse)
print("Validation Average NMSE: ", avg_nmse_overall)

avg_train_mse = np.mean(train_mse_scores, axis=0)
avg_train_nmse = np.sqrt(avg_train_mse) / np.sqrt(np.mean((y_train - y_mean.values)**2, axis=0))
avg_train_nmse_overall = np.mean(avg_train_nmse)
print("Training Average NMSE: ", avg_train_nmse_overall)

# 使用整个训练集重新训练模型
model.fit(X_train, y_train)

# 预测测试集并计算 NMSE
pred = model.predict(X_test)
```

```

mse_test = mean_squared_error(y_test, pred, multioutput='raw_values')
y_test_mean = np.mean(y_test)
nmse_test = np.sqrt(mse_test) / np.sqrt(np.mean((y_test - y_test_mean.values)**2, axis=0))
nmse_test_overall = np.mean(nmse_test)
print("Test NMSE: ", nmse_test_overall)

```

“=====岭回归=====”

```

from sklearn.linear_model import Ridge
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import mean_squared_error
from tqdm.notebook import tqdm
from sklearn.model_selection import KFold
import seaborn as sns
import matplotlib.pyplot as plt

# 使用 KFold 进行交叉验证，并在每次训练过程中使用 tqdm 显示进度
kf = KFold(n_splits=5, shuffle=True, random_state=42)

# 初始化岭回归模型
param_grid = {
    'alpha': [0.001, 0.01, 0.1, 1, 10, 100, 1000] # 超参数选择的范围
}

ridge = Ridge()
model = GridSearchCV(ridge, param_grid, scoring='neg mean squared error', cv=kf, n_jobs=-1,
return_train_score=True)

# 为了可视化，我们不需要再使用 tqdm 展示训练进度
model.fit(X_train, y_train)

# 输出最佳的超参数
print("Best hyperparameters: ", model.best_params_)

# 绘制超参数搜索过程的折线图
plt.figure(figsize=(10, 5))
plt.semilogx(param_grid['alpha'], -model.cv_results_['mean_test_score'], marker='o', label='Validation Score', color='red')
plt.semilogx(param_grid['alpha'], -model.cv_results_['mean_train_score'], marker='o', label='Training Score', color='blue')
plt.xlabel('Alpha')
plt.ylabel('Mean Squared Error')
plt.title('Hyperparameter Search - Ridge Regression')
plt.legend()
plt.grid(True, which='both', linestyle='--', linewidth=0.5)
plt.show()

# 使用训练的模型绘制学习曲线
plot_learning_curve(model, "Learning Curve (Ridge Regression)", X_train, y_train, cv=kf, n_jobs=-1)

# 计算平均的 NMSE
y_mean = np.mean(y_train.values)
avg_mse = -model.best_score_
avg_nmse = np.sqrt(avg_mse) / np.sqrt(np.mean((y_train.values - y_mean)**2))
print("Validation Average NMSE: ", avg_nmse)

```

```

# 使用整个训练集重新训练模型
model.fit(X_train, y_train)

# 预测测试集并计算 NMSE
pred = model.predict(X_test)
mse_test = mean_squared_error(y_test, pred)
y_test_mean = np.mean(y_test.values)
nmse_test = np.sqrt(mse_test) / np.sqrt(np.mean((y_test.values - y_test_mean)**2))
print("Test NMSE: ", nmse_test)

“=====LASSO=====”
from sklearn.linear_model import Lasso
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import mean_squared_error
from tqdm.notebook import tqdm
from sklearn.model_selection import KFold
import seaborn as sns
import matplotlib.pyplot as plt

# 使用 KFold 进行交叉验证，并在每次训练过程中使用 tqdm 显示进度
kf = KFold(n_splits=5, shuffle=True, random_state=42)

# 初始化 Lasso 模型的超参数范围
param_grid = {
    'alpha': [0.001, 0.01, 0.1, 1, 10, 100, 1000] # 超参数选择的范围
}

lasso = Lasso()
model = GridSearchCV(lasso, param_grid, scoring='neg mean squared error', cv=kf, n_jobs=-1,
return_train_score=True)

# 为了可视化，我们不需要再使用 tqdm 展示训练进度
model.fit(X_train, y_train)

# 输出最佳的超参数
print("Best hyperparameters: ", model.best_params_)

# 绘制超参数搜索过程的折线图
plt.figure(figsize=(10, 5))
plt.semilogx(param_grid['alpha'], -model.cv_results_['mean_test_score'], marker='o', label='Validation Score', color='red')
plt.semilogx(param_grid['alpha'], -model.cv_results_['mean_train_score'], marker='o', label='Training Score', color='blue')
plt.xlabel('Alpha')
plt.ylabel('Mean Squared Error')
plt.title('Hyperparameter Search - Lasso Regression')
plt.legend()
plt.grid(True, which='both', linestyle='--', linewidth=0.5)
plt.show()

# 使用训练的模型绘制学习曲线
plot_learning_curve(model, "Learning Curve (Lasso Regression)", X_train, y_train, cv=kf, n_jobs=-1)

# 计算平均的 NMSE
y_mean = np.mean(y_train.values)

```

```

avg_mse = -model.best_score_
avg_nmse = np.sqrt(avg_mse) / np.sqrt(np.mean((y_train.values - y_mean)**2))
print("Validation Average NMSE: ", avg_nmse)

# 使用整个训练集重新训练模型
model.fit(X_train, y_train)

# 预测测试集并计算 NMSE
pred = model.predict(X_test)
mse_test = mean_squared_error(y_test, pred)
y_test_mean = np.mean(y_test.values)
nmse_test = np.sqrt(mse_test) / np.sqrt(np.mean((y_test.values - y_test_mean)**2))
print("Test NMSE: ", nmse_test)

“=====ElasticNet 回归=====”
from sklearn.linear_model import ElasticNet
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import KFold, GridSearchCV
from tqdm.notebook import tqdm
import seaborn as sns
import matplotlib.pyplot as plt

# 使用 KFold 进行交叉验证，并在每次训练过程中使用 tqdm 显示进度
kf = KFold(n_splits=5, shuffle=True, random_state=42)

# 设置 ElasticNet 的参数范围
params = {
    'alpha': [0.001, 0.01, 0.1, 1, 10],
    'l1_ratio': [0.1, 0.3, 0.5, 0.7, 0.9]
}

# 使用 GridSearchCV 搜索最佳参数
model = ElasticNet(max_iter=5000) # 设置较大的迭代次数以确保模型收敛
grid = GridSearchCV(model, params, scoring='neg_mean_squared_error', cv=kf, n_jobs=-1,
return_train_score=True,
verbose=1)

# 训练模型
grid.fit(X_train, y_train)

# 获取结果
results = grid.cv_results_

# 绘制超参数搜索的折线图
plt.figure(figsize=(12, 6))
for l1_ratio in params['l1_ratio']:
    mask = (np.array(results['param_l1_ratio']).data, dtype=float) == l1_ratio
    mean_test_scores = results['mean_test_score'][mask]
    plt.plot(params['alpha'], -mean_test_scores, '-o', label=f'l1_ratio={l1_ratio}')

plt.xscale('log')
plt.xlabel('Alpha')
plt.ylabel('Mean Test NMSE')
plt.legend()
plt.title('Performance vs. Hyperparameters')

```

```

plt.show()

# 绘制交叉验证的热力图
scores_matrix = -results['mean_test_score'].reshape(len(params['l1_ratio']), len(params['alpha']))
plt.figure(figsize=(10, 6))
sns.heatmap(scores_matrix, annot=True, fmt='.5f', cmap='YlGnBu', xticklabels=params['alpha'],
            yticklabels=params['l1_ratio'])
plt.xlabel('Alpha')
plt.ylabel('L1 Ratio')
plt.title('Hyperparameters GridSearch Heatmap')
plt.show()

# 使用训练的模型绘制学习曲线
best_model = grid.best_estimator_
plot_learning_curve(best_model, "Learning Curve (ElasticNet Regression)", X_train, y_train, cv=kf,
                    n_jobs=-1)

# 使用整个训练集重新训练模型
best_model.fit(X_train, y_train)

# 预测测试集并计算 NMSE
pred = best_model.predict(X_test)
mse_test = mean_squared_error(y_test.sum(axis=1), pred.sum(axis=1))
y_test_mean = np.mean(y_test.sum(axis=1))
nmse_test = np.sqrt(mse_test / np.mean((y_test.sum(axis=1) - y_test_mean) ** 2))
print("Test NMSE: ", nmse_test)

“=====KNN=====”
from sklearn.neighbors import KNeighborsRegressor
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import KFold, GridSearchCV
from tqdm.notebook import tqdm
import seaborn as sns
import matplotlib.pyplot as plt

# 使用 KFold 进行交叉验证，并在每次训练过程中使用 tqdm 显示进度
kf = KFold(n_splits=5, shuffle=True, random_state=42)

# 设置 KNeighborsRegressor 的参数范围
params = {
    'n_neighbors': [3, 5, 7, 9, 11],
    'weights': ['uniform', 'distance']
}

# 使用 GridSearchCV 搜索最佳参数
model = KNeighborsRegressor()
grid = GridSearchCV(model, params, scoring='neg mean squared error', cv=kf, n_jobs=-1,
                    return_train_score=True, verbose=1)

# 训练模型
grid.fit(X_train, y_train)

# 获取结果
results = grid.cv_results_

```

```

# 绘制超参数搜索的折线图
plt.figure(figsize=(12, 6))
for weight in params['weights']:
    mask = (np.array(results['param_weights'].data) == weight)
    mean_test_scores = results['mean_test_score'][mask]
    plt.plot(params['n_neighbors'], -mean_test_scores, '-o', label=f'weights={weight}')

plt.xlabel('Number of Neighbors')
plt.ylabel('Mean Test NMSE')
plt.legend()
plt.title('Performance vs. Hyperparameters')
plt.show()

# 绘制交叉验证的热力图
scores_matrix = -results['mean_test_score'].reshape(len(params['weights']), len(params['n_neighbors']))
plt.figure(figsize=(10, 6))
sns.heatmap(scores_matrix, annot=True, fmt='.5f', cmap='YlGnBu', xticklabels=params['n_neighbors'],
            yticklabels=params['weights'])
plt.xlabel('Number of Neighbors')
plt.ylabel('Weights')
plt.title('Hyperparameters GridSearch Heatmap')
plt.show()

# 使用训练的模型绘制学习曲线
best_model = grid.best_estimator
plot_learning_curve(best_model, "Learning Curve (KNN Regression)", X_train, y_train, cv=kf, n_jobs=-1)

# 使用整个训练集重新训练模型
best_model.fit(X_train, y_train)

# 预测测试集并计算 NMSE
pred = best_model.predict(X_test)
mse_test = mean_squared_error(y_test.sum(axis=1), pred.sum(axis=1))
y_test_mean = np.mean(y_test.sum(axis=1))
nmse_test = np.sqrt(mse_test / np.mean((y_test.sum(axis=1) - y_test_mean) ** 2))
print("Test NMSE: ", nmse_test)

```

“=====回归决策树=====”

```

from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import KFold, GridSearchCV
from tqdm.notebook import tqdm, notebook as tqdm
import seaborn as sns
import matplotlib.pyplot as plt

# 使用 KFold 进行交叉验证
kf = KFold(n_splits=5, shuffle=True, random_state=42)

# 设置 DecisionTreeRegressor 的参数范围
params = {
    'max_depth': [3, 5, 7, 10, None],
    'min_samples_split': [2, 5, 10]
}

```

```

# 使用 GridSearchCV 搜索最佳参数
model = DecisionTreeRegressor(random_state=42)
grid = GridSearchCV(model, params, scoring='neg mean squared error', cv=kf, n_jobs=-1,
return_train_score=True,
                      verbose=1)

# 使用 tqdm 展示模型训练过程
with tqdm(total=len(params['max_depth']) * len(params['min_samples_split']) * kf.get_n_splits(),
          desc="GridSearchCV") as pbar:
    def progress_callback(*args, **kwargs):
        pbar.update(1)

    # 使用 progress_callback 更新进度
    grid._update_progress = progress_callback

    # 训练模型
    grid.fit(X_train, y_train)

# 获取结果
results = grid.cv_results_

# 绘制超参数搜索的折线图
plt.figure(figsize=(12, 6))
for depth in params['max_depth']:
    mask = (np.array(results['param_max_depth'].data) == depth)
    mean_test_scores = results['mean_test_score'][mask]
    plt.plot(params['min_samples_split'], -mean_test_scores, '-o', label=f'max_depth={depth}')

plt.xlabel('Min Samples Split')
plt.ylabel('Mean Test NMSE')
plt.legend()
plt.title('Performance vs. Hyperparameters')
plt.show()

# 绘制交叉验证的热力图
scores_matrix = -results['mean_test_score'].reshape(len(params['min_samples_split']),
len(params['max_depth']))
plt.figure(figsize=(10, 6))
sns.heatmap(scores_matrix, annot=True, fmt='.5f', cmap='YlGnBu', xticklabels=params['max_depth'],
            yticklabels=params['min_samples_split'])
plt.xlabel('Max Depth')
plt.ylabel('Min Samples Split')
plt.title('Hyperparameters GridSearch Heatmap')
plt.show()

# 使用训练的模型绘制学习曲线
best_model = grid.best_estimator
plot_learning_curve(best_model, "Learning Curve (Decision Tree Regression)", X_train, y_train, cv=kf,
n_jobs=-1)

# 使用整个训练集重新训练模型
best_model.fit(X_train, y_train)

# 预测测试集并计算 NMSE
pred = best_model.predict(X_test)

```

```

mse_test = mean_squared_error(y_test.sum(axis=1), pred.sum(axis=1))
y_test_mean = np.mean(y_test.sum(axis=1))
nmse_test = np.sqrt(mse_test / np.mean((y_test.sum(axis=1) - y_test_mean) ** 2))
print("Test NMSE: ", nmse_test)

```

“=====随机森林=====”

```

from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import KFold, ParameterGrid
import seaborn as sns
import matplotlib.pyplot as plt
from tqdm.notebook import tqdm notebook

# 使用 KFold 进行交叉验证
kf = KFold(n_splits=5, shuffle=True, random_state=42)

# 设置 RandomForestRegressor 的参数范围
params = {
    'n_estimators': [50, 100, 150],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10]
}

# 初始化变量来存储最佳模型和最佳得分
best_model = None
best_score = float('-inf')
all_results = []

# 使用 tqdm_notebook 显示训练进度
param_combinations = list(ParameterGrid(params))
for param_combination in tqdm_notebook(param_combinations, desc="Training..."):
    model = RandomForestRegressor(**param_combination, random_state=42)
    scores = []
    for train_idx, test_idx in kf.split(X_train):
        model.fit(X_train.iloc[train_idx], y_train.iloc[train_idx])
        score = model.score(X_train.iloc[test_idx], y_train.iloc[test_idx])
        scores.append(score)
    mean_score = np.mean(scores)
    all_results.append(mean_score)

    if mean_score > best_score:
        best_score = mean_score
        best_model = model

# 绘制超参数搜索的折线图
# 这里只展示 n_estimators 参数对结果的影响，因为展示所有参数会使图形复杂
plt.figure(figsize=(12, 6))
plt.plot(params['n_estimators'], all_results[:len(params['n_estimators'])], '-o')
plt.xlabel('Number of Estimators')
plt.ylabel('Mean Test Score')
plt.title('Performance vs. Number of Estimators')
plt.show()

# 使用训练的模型绘制学习曲线
plot_learning_curve(best_model, "Learning Curve (Random Forest Regression)", X_train, y_train, cv=kf,

```

```

n_jobs=-1)

# 使用整个训练集重新训练模型
best_model.fit(X_train, y_train)

# 预测测试集并计算 NMSE
pred = best_model.predict(X_test)
mse_test = mean_squared_error(y_test.sum(axis=1), pred.sum(axis=1))
y_test_mean = np.mean(y_test.sum(axis=1))
nmse_test = np.sqrt(mse_test / np.mean((y_test.sum(axis=1) - y_test_mean) ** 2))
print("Test NMSE: ", nmse_test)

“=====AdaBoost=====”
from sklearn.ensemble import AdaBoostRegressor
from sklearn.multioutput import MultiOutputRegressor
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import KFold, ParameterGrid
from tqdm.notebook import tqdm
import seaborn as sns
import matplotlib.pyplot as plt

# 使用 KFold 进行交叉验证
kf = KFold(n_splits=5, shuffle=True, random_state=42)

# 设置 AdaBoostRegressor 的参数范围
params = {
    'n_estimators': [50, 100, 200],
    'learning_rate': [0.01, 0.1, 1]
}

# 手动搜索最佳参数
best_score = float('inf')
best_params = None
all_scores = []

param_grid = list(ParameterGrid(params))

with tqdm(total=len(param_grid) * kf.get_n_splits(), desc="GridSearchCV") as pbar:
    for param_combination in param_grid:
        fold_scores = []
        for train_idx, test_idx in kf.split(X_train):
            base_model = AdaBoostRegressor(random_state=42, **param_combination)
            model = MultiOutputRegressor(base_model) # 将 AdaBoostRegressor 包装为多输出回
            归器

            X_train_fold, X_test_fold = X_train.iloc[train_idx], X_train.iloc[test_idx]
            y_train_fold, y_test_fold = y_train.iloc[train_idx], y_train.iloc[test_idx]

            model.fit(X_train_fold, y_train_fold)
            pred = model.predict(X_test_fold)

            mse = mean_squared_error(y_test_fold, pred)
            fold_scores.append(mse)

        pbar.update(1)

```

```

    mean_score = np.mean(fold_scores)
    all_scores.append(mean_score)

    if mean_score < best_score:
        best_score = mean_score
        best_params = param_combination

# 绘制交叉验证的热力图
scores_matrix = np.array(all_scores).reshape(len(params['learning_rate']), len(params['n_estimators']))
plt.figure(figsize=(10, 6))
sns.heatmap(scores_matrix, annot=True, fmt='.5f', cmap='YlGnBu', xticklabels=params['n_estimators'],
            yticklabels=params['learning_rate'])
plt.xlabel('n_estimators')
plt.ylabel('learning_rate')
plt.title('Hyperparameters GridSearch Heatmap')
plt.show()

# 使用训练的模型绘制学习曲线
base_best_model = AdaBoostRegressor(random_state=42, **best_params)
best_model = MultiOutputRegressor(base_best_model)
plot_learning_curve(best_model, "Learning Curve (AdaBoost Regression)", X_train, y_train, cv=kf,
                    n_jobs=-1)

# 使用整个训练集重新训练模型
best_model.fit(X_train, y_train)

# 预测测试集并计算 NMSE
pred = best_model.predict(X_test)
mse_test = mean_squared_error(y_test.sum(axis=1), pred.sum(axis=1))
y_test_mean = np.mean(y_test.sum(axis=1))
nmse_test = np.sqrt(mse_test / np.mean((y_test.sum(axis=1) - y_test_mean) ** 2))
print("Test NMSE: ", nmse_test)

“=====XGBoost=====”
from sklearn.model_selection import ParameterGrid
import xgboost as xgb
from sklearn.multioutput import MultiOutputRegressor
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import KFold
from tqdm.notebook import tqdm
import seaborn as sns
import matplotlib.pyplot as plt

# 使用 KFold 进行交叉验证
kf = KFold(n_splits=5, shuffle=True, random_state=42)

# 设置 XGBoost 的参数范围
params = {
    'n_estimators': [50, 100, 200],
    'learning_rate': [0.01, 0.1, 1],
    'max_depth': [3, 6, 9]
}

# 手动搜索最佳参数
best_score = float('inf')
best_params = None

```

```

all_scores = []

param_grid = list(ParameterGrid(params))

with tqdm(total=len(param_grid) * kf.get_n_splits(), desc="GridSearchCV") as pbar:
    for param_combination in param_grid:
        fold_scores = []
        for train_idx, test_idx in kf.split(X_train):
            base_model = xgb.XGBRegressor(objective='reg:squarederror', random_state=42,
**param_combination)
            model = MultiOutputRegressor(base_model) # 将 XGBoost 包装为多输出回归器

            X_train_fold, X_test_fold = X_train.iloc[train_idx], X_train.iloc[test_idx]
            y_train_fold, y_test_fold = y_train.iloc[train_idx], y_train.iloc[test_idx]

            model.fit(X_train_fold, y_train_fold)
            pred = model.predict(X_test_fold)

            mse = mean_squared_error(y_test_fold, pred)
            fold_scores.append(mse)

        pbar.update(1)

    mean_score = np.mean(fold_scores)
    all_scores.append(mean_score)

    if mean_score < best_score:
        best_score = mean_score
        best_params = param_combination

# 绘制交叉验证的热力图
scores_matrix = np.array(all_scores).reshape(len(params['learning_rate']), len(params['n_estimators']),
len(params['max_depth'])).mean(axis=2)

plt.figure(figsize=(10, 6))
sns.heatmap(scores_matrix, annot=True, fmt='.5f', cmap='YlGnBu', xticklabels=params['n_estimators'],
yticklabels=params['learning_rate'])
plt.xlabel('n_estimators')
plt.ylabel('learning_rate')
plt.title('Hyperparameters GridSearch Heatmap')
plt.show()

# 使用训练的模型绘制学习曲线
base_best_model = xgb.XGBRegressor(objective='reg:squarederror', random_state=42, **best_params)
best_model = MultiOutputRegressor(base_best_model)
plot_learning_curve(best_model, "Learning Curve (XGBoost Regression)", X_train, y_train, cv=kf,
n_jobs=-1)

# 使用整个训练集重新训练模型
best_model.fit(X_train, y_train)

# 预测测试集并计算 NMSE
pred = best_model.predict(X_test)
mse_test = mean_squared_error(y_test.sum(axis=1), pred.sum(axis=1))
y_test_mean = np.mean(y_test.sum(axis=1))
nmse_test = np.sqrt(mse_test / np.mean((y_test.sum(axis=1) - y_test_mean) ** 2))
print("Test NMSE: ", nmse_test)

```

```

“=====CatBoost=====”

from catboost import CatBoostRegressor
from sklearn.multioutput import MultiOutputRegressor
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import KFold
from tqdm.notebook import tqdm
import seaborn as sns
import matplotlib.pyplot as plt

# 使用 KFold 进行交叉验证
kf = KFold(n_splits=5, shuffle=True, random_state=42)

# 设置 CatBoostRegressor 的参数范围
params = {
    'iterations': [500, 1000],
    'learning_rate': [0.01, 0.1],
    'depth': [4, 6, 8]
}

# 手动搜索最佳参数
best_score = float('inf')
best_params = None
all_scores = []

param_grid = list(ParameterGrid(params))

with tqdm(total=len(param_grid) * kf.get_n_splits(), desc="GridSearchCV") as pbar:
    for param_combination in param_grid:
        fold_scores = []
        for train_idx, test_idx in kf.split(X_train):
            base_model = CatBoostRegressor(random_state=42, verbose=0, **param_combination)
            model = MultiOutputRegressor(base_model) # 将 CatBoostRegressor 包装为多输出回归器

            X_train_fold, X_test_fold = X_train.iloc[train_idx], X_train.iloc[test_idx]
            y_train_fold, y_test_fold = y_train.iloc[train_idx], y_train.iloc[test_idx]

            model.fit(X_train_fold, y_train_fold)
            pred = model.predict(X_test_fold)

            mse = mean_squared_error(y_test_fold, pred)
            fold_scores.append(mse)

        pbar.update(1)

    mean_score = np.mean(fold_scores)
    all_scores.append(mean_score)

    if mean_score < best_score:
        best_score = mean_score
        best_params = param_combination

# 绘制交叉验证的热力图
scores_matrix = np.array(all_scores).reshape(len(params['learning_rate']), len(params['iterations']),
                                              len(params['depth'])).mean(axis=2)

plt.figure(figsize=(10, 6))

```

```
sns.heatmap(scores_matrix, annot=True, fmt='.5f', cmap='YlGnBu', xticklabels=params['iterations'],
            yticklabels=params['learning_rate'])
plt.xlabel('iterations')
plt.ylabel('learning_rate')
plt.title('Hyperparameters GridSearch Heatmap')
plt.show()
```

使用训练的模型绘制学习曲线

```
base_best_model = CatBoostRegressor(random_state=42, **best_params, verbose=0)
best_model = MultiOutputRegressor(base_best_model)
plot_learning_curve(best_model, "Learning Curve (CatBoost Regression)", X_train, y_train, cv=kf,
                    n_jobs=-1)
```

使用整个训练集重新训练模型

```
best_model.fit(X_train, y_train)
```

预测测试集并计算 NMSE

```
pred = best_model.predict(X_test)
mse_test = mean_squared_error(y_test.sum(axis=1), pred.sum(axis=1))
y_test_mean = np.mean(y_test.sum(axis=1))
nmse_test = np.sqrt(mse_test / np.mean((y_test.sum(axis=1) - y_test_mean) ** 2))
print("Test NMSE: ", nmse_test)
```

“=====GradientBoosting=====”

```
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.multioutput import MultiOutputRegressor
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import KFold, ParameterGrid
from tqdm.notebook import tqdm
import seaborn as sns
import matplotlib.pyplot as plt
```

使用 KFold 进行交叉验证

```
kf = KFold(n_splits=5, shuffle=True, random_state=42)
```

设置 GradientBoostingRegressor 的参数范围

```
params = {
    'n_estimators': [50, 100, 200],
    'learning_rate': [0.01, 0.1, 1],
    'max_depth': [3, 4, 5]
}
```

手动搜索最佳参数

```
best_score = float('inf')
best_params = None
all_scores = []
```

```
param_grid = list(ParameterGrid(params))
```

```
with tqdm(total=len(param_grid) * kf.get_n_splits(), desc="GridSearchCV") as pbar:
```

```
    for param_combination in param_grid:
```

```
        fold_scores = []
```

```
        for train_idx, test_idx in kf.split(X_train):
```

```
            base_model = GradientBoostingRegressor(random_state=42, **param_combination)
```

```
            model = MultiOutputRegressor(base_model) # 将 GradientBoostingRegressor 包装为多
```

```

X_train_fold, X_test_fold = X_train.iloc[train_idx], X_train.iloc[test_idx]
y_train_fold, y_test_fold = y_train.iloc[train_idx], y_train.iloc[test_idx]

model.fit(X_train_fold, y_train_fold)
pred = model.predict(X_test_fold)

mse = mean_squared_error(y_test_fold, pred)
fold_scores.append(mse)

pbar.update(1)

mean_score = np.mean(fold_scores)
all_scores.append(mean_score)

if mean_score < best_score:
    best_score = mean_score
    best_params = param_combination

# 绘制交叉验证的热力图
scores_matrix = np.array(all_scores).reshape(len(params['learning_rate']), len(params['n_estimators']),
                                              len(params['max_depth']).mean(axis=2) # 对于
三个超参数，我们在这里仅选择了两个来绘制热力图
plt.figure(figsize=(10, 6))
sns.heatmap(scores_matrix, annot=True, fmt='.5f', cmap='YlGnBu', xticklabels=params['n_estimators'],
            yticklabels=params['learning_rate'])
plt.xlabel('n_estimators')
plt.ylabel('learning_rate')
plt.title('Hyperparameters GridSearch Heatmap')
plt.show()

# 使用训练的模型绘制学习曲线
base_best_model = GradientBoostingRegressor(random_state=42, **best_params)
best_model = MultiOutputRegressor(base_best_model)
plot_learning_curve(best_model, "Learning Curve (GradientBoosting Regression)", X_train, y_train, cv=kf,
                    n_jobs=-1)

# 使用整个训练集重新训练模型
best_model.fit(X_train, y_train)

# 预测测试集并计算 NMSE
pred = best_model.predict(X_test)
mse_test = mean_squared_error(y_test.sum(axis=1), pred.sum(axis=1))
y_test_mean = np.mean(y_test.sum(axis=1))
nmse_test = np.sqrt(mse_test / np.mean((y_test.sum(axis=1) - y_test_mean) ** 2))
print("Test NMSE: ", nmse_test)

```

“=====BP 神经网络=====”

```

import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader, TensorDataset
from tqdm.notebook import tqdm
import matplotlib.pyplot as plt

```

```
import seaborn as sns
```

```
# 定义 BP 神经网络模型
```

```
class BPNN(nn.Module):
    def __init__(self, input_dim, hidden_dim, output_dim):
        super(BPNN, self).__init__()
        self.fc1 = nn.Linear(input_dim, hidden_dim)
        self.fc2 = nn.Linear(hidden_dim, hidden_dim)
        self.fc3 = nn.Linear(hidden_dim, output_dim)
        self.leakyrelu = nn.LeakyReLU(negative_slope=0.01) # 使用 LeakyReLU

    def forward(self, x):
        x = self.leakyrelu(self.fc1(x))
        x = self.leakyrelu(self.fc2(x))
        x = self.fc3(x)
        return x
```

```
# 计算整体的 NNBE
```

```
def calculate_nnbe(y_true, y_pred):
    numerator = torch.sum((y_true - y_pred) ** 2)
    denominator = torch.sum((y_true - torch.mean(y_true)) ** 2)
    return torch.sqrt(numerator / denominator).item()
```

```
# 超参数设置
```

```
learning_rate = 0.001
epochs = 500
hidden_dim = 128
batch_size = 32
```

```
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
```

```
# 数据转换为 PyTorch 格式
```

```
train_dataset = TensorDataset(torch.tensor(X_train.values).float(), torch.tensor(y_train.values).float())
train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True)
```

```
test_dataset = TensorDataset(torch.tensor(X_test.values).float(), torch.tensor(y_test.values).float())
test_loader = DataLoader(test_dataset, batch_size=batch_size, shuffle=False)
```

```
# 初始化模型
```

```
model = BPNN(X_train.shape[1], hidden_dim, y_train.shape[1]).to(device)
criterion = nn.MSELoss()
optimizer = optim.Adam(model.parameters(), lr=learning_rate)
```

```
# 模型训练
```

```
train_nbes = []
validation_nbes = []
```

```
for epoch in tqdm(range(epochs), desc="Training model"):
    model.train()
    train_loss = 0.0
    for X_batch, y_batch in train_loader:
        X_batch, y_batch = X_batch.to(device), y_batch.to(device)

        optimizer.zero_grad()
```

```

        outputs = model(X_batch)
        loss = criterion(outputs, y_batch)
        loss.backward()
        optimizer.step()

    train_loss += loss.item() * X_batch.size(0)

train_nnbe = calculate_nnbe(torch.tensor(y_train.values).float().to(device),
                             model(torch.tensor(X_train.values).float().to(device)))
train_nnbes.append(train_nnbe)

model.eval()
validation_loss = 0.0
with torch.no_grad():
    for X_batch, y_batch in test_loader:
        X_batch, y_batch = X_batch.to(device), y_batch.to(device)

        outputs = model(X_batch)
        loss = criterion(outputs, y_batch)
        validation_loss += loss.item() * X_batch.size(0)

validation_nnbe = calculate_nnbe(torch.tensor(y_test.values).float().to(device),
                                 model(torch.tensor(X_test.values).float().to(device)))
validation_nnbes.append(validation_nnbe)

# 绘制训练 NNBE 和验证 NNBE
plt.figure(figsize=(12, 6))
plt.plot(train_nnbes, label="Training NNBE", color=sns.color_palette("husl", 8)[3])
plt.plot(validation_nnbes, label="Validation NNBE", color=sns.color_palette("husl", 8)[4])
plt.title("Training and Validation NNBE", fontsize=20)
plt.xlabel("Epoch", fontsize=16)
plt.ylabel("NNBE", fontsize=16)
plt.legend(fontsize=14)
plt.grid(color='gray', linestyle='--', linewidth=0.5)
plt.tight_layout()
plt.show()

# 计算测试集的 NNBE
y_pred = model(torch.tensor(X_test.values).float().to(device))
nnbe = calculate_nnbe(torch.tensor(y_test.values).float().to(device), y_pred)
print(f"Test NNBE: {nnbe:.4f}")

```

“=====LSTM=====”

```

import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader, TensorDataset
from tqdm.notebook import tqdm
import matplotlib.pyplot as plt
import seaborn as sns

# LSTM 模型定义
class LSTMRegressor(nn.Module):
    def __init__(self, input_dim, hidden_dim, output_dim, num_layers):
        super(LSTMRegressor, self).__init__()

```

```

self.lstm = nn.LSTM(input_dim, hidden_dim, num_layers, batch_first=True)
self.decoder = nn.Linear(hidden_dim, output_dim)

def forward(self, x):
    h0 = torch.zeros(num_layers, x.size(0), hidden_dim).to(device)
    c0 = torch.zeros(num_layers, x.size(0), hidden_dim).to(device)
    out, _ = self.lstm(x, (h0, c0))
    out = self.decoder(out[:, -1, :])
    return out

# NNBE loss function
def nnbe_loss(y_pred, y_true):
    numerator = torch.sum((y_true - y_pred) ** 2)
    denominator = torch.sum((y_true - torch.mean(y_true)) ** 2)
    return torch.sqrt(numerator / denominator)

# 超参数设置
learning_rate = 0.005
epochs = 1000
hidden_dim = 128
num_layers = 2
batch_size = 32

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

# 初始化模型
model = LSTMRegressor(X_train.shape[1], hidden_dim, y_train.shape[1], num_layers).to(device)
optimizer = optim.Adam(model.parameters(), lr=learning_rate)

# 模型训练
train_losses = []
test_losses = []

for epoch in tqdm(range(epochs), desc="Training model"):
    model.train()
    train_loss = 0.0
    for X_batch, y_batch in train_loader:
        X_batch, y_batch = X_batch.to(device), y_batch.to(device)

        optimizer.zero_grad()
        outputs = model(X_batch)
        loss = nnbe_loss(outputs, y_batch)
        loss.backward()
        optimizer.step()

        train_loss += loss.item() * X_batch.size(0)

    train_losses.append(train_loss / len(train_loader.dataset))

    model.eval()
    test_loss = 0.0
    with torch.no_grad():
        for X_batch, y_batch in test_loader:
            X_batch, y_batch = X_batch.to(device), y_batch.to(device)

```

```

        outputs = model(X_batch)
        loss = nnbe_loss(outputs, y_batch)
        test_loss += loss.item() * X_batch.size(0)
    test_losses.append(test_loss / len(test_loader.dataset))

# 绘制训练损失和测试损失曲线
plt.figure(figsize=(12, 6))
plt.plot(train_losses, label="Training Loss", color=sns.color_palette("husl", 8)[3])
plt.plot(test_losses, label="Validation Loss", color=sns.color_palette("husl", 8)[4])
plt.title("Training and Validation Loss", fontsize=20)
plt.xlabel("Epoch", fontsize=16)
plt.ylabel("NNBE", fontsize=16)
plt.legend(fontsize=14)
plt.grid(color='gray', linestyle='--', linewidth=0.5)
plt.tight_layout()
plt.show()

# 计算测试集的 NMSE
y_pred = model(torch.tensor(X_test.values).float().unsqueeze(1).to(device))
nnbe = nnbe_loss(y_pred, torch.tensor(y_test.values).float().to(device)).item()

print(f"Training Average NMSE: {sum(train_losses) / len(train_losses):.4f}")
print(f"Validation Average NMSE: {sum(test_losses) / len(test_losses):.4f}")
print(f"Test NMSE: {nnbe:.4f}")

```

“=====GRU=====”

```

import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader, TensorDataset
from tqdm.notebook import tqdm
import matplotlib.pyplot as plt
import seaborn as sns

# GRU 模型定义
class GRURegressor(nn.Module):
    def __init__(self, input_dim, hidden_dim, output_dim, num_layers):
        super(GRURegressor, self).__init__()

        self.gru = nn.GRU(input_dim, hidden_dim, num_layers, batch_first=True)
        self.decoder = nn.Linear(hidden_dim, output_dim)

    def forward(self, x):
        out, _ = self.gru(x)
        out = self.decoder(out[:, -1, :]) # 使用最后一个时间步的输出作为输出
        return out

# NNBE loss function
def nnbe_loss(y_pred, y_true):
    numerator = torch.sum((y_true - y_pred) ** 2)
    denominator = torch.sum((y_true - torch.mean(y_true)) ** 2)
    return torch.sqrt(numerator / denominator)

```

```

# 超参数设置
learning_rate = 0.005
epochs = 500
hidden_dim = 128
num_layers = 2
batch_size = 32

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

# 初始化模型
model = GRURegressor(X_train.shape[1], hidden_dim, y_train.shape[1], num_layers).to(device)
optimizer = optim.Adam(model.parameters(), lr=learning_rate)

# 模型训练
train_losses = []
test_losses = []

for epoch in tqdm(range(epochs), desc="Training model"):
    model.train()
    train_loss = 0.0
    for X_batch, y_batch in train_loader:
        X_batch, y_batch = X_batch.to(device), y_batch.to(device)

        optimizer.zero_grad()
        outputs = model(X_batch)
        loss = nnbe_loss(outputs, y_batch)
        loss.backward()
        torch.nn.utils.clip_grad_norm_(model.parameters(), 1.0)
        optimizer.step()

        train_loss += loss.item() * X_batch.size(0)
    train_losses.append(train_loss / len(train_loader.dataset))

    model.eval()
    test_loss = 0.0
    with torch.no_grad():
        for X_batch, y_batch in test_loader:
            X_batch, y_batch = X_batch.to(device), y_batch.to(device)
            outputs = model(X_batch)
            loss = nnbe_loss(outputs, y_batch)
            test_loss += loss.item() * X_batch.size(0)
        test_losses.append(test_loss / len(test_loader.dataset))

# 绘制训练损失和测试损失曲线
plt.figure(figsize=(12, 6))
plt.plot(train_losses, label="Training Loss", color=sns.color_palette("husl", 8)[3])
plt.plot(test_losses, label="Validation Loss", color=sns.color_palette("husl", 8)[4])
plt.title("Training and Validation Loss", fontsize=20)
plt.xlabel("Epoch", fontsize=16)
plt.ylabel("NNBE", fontsize=16)
plt.legend(fontsize=14)
plt.grid(color='gray', linestyle='--', linewidth=0.5)
plt.tight_layout()
plt.show()

# 计算测试集的 NMSE

```

```

y_pred = model(torch.tensor(X_test.values).float()).unsqueeze(1).to(device)
nnbe = nnbe_loss(y_pred, torch.tensor(y_test.values).float().to(device)).item()

print(f'Training Average NMSE: {sum(train_losses) / len(train_losses):.4f}')
print(f'Validation Average NMSE: {sum(test_losses) / len(test_losses):.4f}')
print(f'Test NMSE: {nnbe:.4f}')

```

“=====Transformer=====”

```

import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader, TensorDataset
from tqdm.notebook import tqdm
import matplotlib.pyplot as plt
import seaborn as sns

# Transformer 模型定义
class TransformerRegressor(nn.Module):
    def __init__(self, input_dim, hidden_dim, output_dim, nhead, num_layers):
        super(TransformerRegressor, self).__init__()

        self.embedding = nn.Linear(input_dim, hidden_dim)
        self.layer_norm = nn.LayerNorm(hidden_dim)
        self.transformer = nn.Transformer(hidden_dim, nhead, num_layers)
        self.decoder = nn.Linear(hidden_dim, output_dim)

    def forward(self, x):
        x = self.embedding(x)
        x = self.layer_norm(x)
        x = self.transformer(x, x)
        x = self.decoder(x)
        return x.squeeze(1)

# NNBE loss function
def nnbe_loss(y_pred, y_true):
    numerator = torch.sum((y_true - y_pred) ** 2)
    denominator = torch.sum((y_true - torch.mean(y_true)) ** 2)
    return torch.sqrt(numerator / denominator)

# 超参数设置
learning_rate = 0.005
epochs = 100
hidden_dim = 128
nhead = 2
num_layers = 2
batch_size = 32

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

# 数据转换为 PyTorch 格式并添加一个虚拟的序列长度维度
train_dataset = TensorDataset(torch.tensor(X_train.values).float().unsqueeze(1),
                                torch.tensor(y_train.values).float())
train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True)

```

```

test_dataset = TensorDataset(torch.tensor(X_test.values).float().unsqueeze(1),
torch.tensor(y_test.values).float())
test_loader = DataLoader(test_dataset, batch_size=batch_size, shuffle=False)

# 初始化模型
model = TransformerRegressor(X_train.shape[1], hidden_dim, y_train.shape[1], nhead,
num_layers).to(device)
optimizer = optim.Adam(model.parameters(), lr=learning_rate)

# 模型训练
train_losses = []
test_losses = []

for epoch in tqdm(range(epochs), desc="Training model"):
    model.train()
    train_loss = 0.0
    for X_batch, y_batch in train_loader:
        X_batch, y_batch = X_batch.to(device), y_batch.to(device)

        optimizer.zero_grad()
        outputs = model(X_batch)
        loss = nnbe_loss(outputs, y_batch)
        loss.backward()
        torch.nn.utils.clip_grad_norm_(model.parameters(), 1.0)
        optimizer.step()

        train_loss += loss.item() * X_batch.size(0)

    train_losses.append(train_loss / len(train_loader.dataset))

    model.eval()
    test_loss = 0.0
    with torch.no_grad():
        for X_batch, y_batch in test_loader:
            X_batch, y_batch = X_batch.to(device), y_batch.to(device)
            outputs = model(X_batch)
            loss = nnbe_loss(outputs, y_batch)
            test_loss += loss.item() * X_batch.size(0)
        test_losses.append(test_loss / len(test_loader.dataset))

# 绘制训练损失和测试损失曲线
plt.figure(figsize=(12, 6))
plt.plot(train_losses, label="Training Loss", color=sns.color_palette("husl", 8)[3])
plt.plot(test_losses, label="Validation Loss", color=sns.color_palette("husl", 8)[4])
plt.title("Training and Validation Loss", fontsize=20)
plt.xlabel("Epoch", fontsize=16)
plt.ylabel("NNBE", fontsize=16)
plt.legend(fontsize=14)
plt.grid(color='gray', linestyle='--', linewidth=0.5)
plt.tight_layout()
plt.show()

# 计算测试集的 NMSE
y_pred = model(torch.tensor(X_test.values).float().unsqueeze(1).to(device))
nnbe = nnbe_loss(y_pred, torch.tensor(y_test.values).float().to(device)).item()

```

```
print(f"Training Average NMSE: {sum(train_losses) / len(train_losses):.4f}")
print(f"Validation Average NMSE: {sum(test_losses) / len(test_losses):.4f}")
print(f"Test NMSE: {nnbe:.4f}")
```

“=====推荐系统=====”

“=====读取数据及其预处理=====”

```
import pandas as pd
import numpy as np
from glob import glob

# 定义路径
app_class_path = "C:/Users/Desktop/A/app_class.csv"
days_data_path = "C:/Users/Desktop/A/day*.txt"

# 读取 app 类别数据
app_class = pd.read_csv(app_class_path, header=None, names=["appid", "app_class"], dtype={"appid": str,
"app_class": str})

# 删除 appid 列中包含非数字字符的行
app_class = app_class[app_class["appid"].str.isnumeric()]

# 数据类型转换以节省内存
dtypes = {
    "uid": str,
    "appid": str,
    "app_type": "category",
    "start_day": np.int8,
    "start_time": str,
    "end_day": np.int8,
    "end_time": str,
    "duration": np.int32,
    "up_flow": np.int32,
    "down_flow": np.int32,
}

data = pd.DataFrame() # 初始化一个空的 data DataFrame 来存储合并的数据
chunk_size = 500000 # 设置块大小

# 从第一天的数据文件中抽取 1 万用户
first_day_file = "C:/Users/Desktop/A/day01.txt"
first_day_data = pd.read_csv(first_day_file, header=None, names=["uid", "appid", "app_type", "start_day",
"start_time", "end_day", "end_time", "duration", "up_flow", "down_flow"], dtype=dtypes)
selected_uids = set(first_day_data["uid"].sample(n=20000, replace=False))

for file in glob(days_data_path):
    for chunk in pd.read_csv(file, header=None, names=["uid", "appid", "app_type", "start_day",
"start_time", "end_day", "end_time", "duration", "up_flow", "down_flow"], dtype=dtypes,
chunksize=chunk_size):
        # 只保留已选择的用户 ID 的数据
        chunk = chunk[chunk["uid"].isin(selected_uids)]

        # 删除 appid 列中包含非数字字符的行
        chunk = chunk[chunk["appid"].str.isnumeric()]
```

```

# 与 app_class 进行合并
chunk = pd.merge(chunk, app_class, on="appid", how="left")

# 追加到 data DataFrame 中
data = pd.concat([data, chunk], ignore_index=True)

# 释放内存
del chunk

# 筛选出用户数超过 10 的 APP
apps_counts = data['appid'].value_counts()
apps_to_keep = apps_counts[apps_counts > 10].index
data = data[data['appid'].isin(apps_to_keep)]

# 剔除 duration、up_flow 和 down_flow 为 0 的行
data = data[(data["duration"] > 0) & (data["up_flow"] > 0) & (data["down_flow"] > 0)]

# 剔除使用时长过于短（小于 5 秒）的行
data = data[data["duration"] >= 5]

# 对于未知类别的 APP，我们将其类别标记为"unknown"
data["app_class"] = data["app_class"].fillna("unknown")

# 计算评分
data["score"] = data["duration"] + data["up_flow"] + data["down_flow"]
# 归一化评分到 1-5 范围
data["score"] = ((data["score"] - data["score"].min()) / (data["score"].max() - data["score"].min()) * 4) + 1

# 将数据分为训练集和测试集
train_data = data[data["start_day"] <= 15]
test_data = data[data["start_day"] > 15]

# 将数据转为推荐系统需要的格式：用户 ID、物品 ID 和评分
train_data = train_data[["uid", "appid", "score"]]
test_data = test_data[["uid", "appid", "score"]]

```

“=====2.1 基于内容的推荐(Content-based recommendation)=====”

```

import matplotlib.pyplot as plt
from tqdm.notebook import tqdm
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.metrics import mean_absolute_error, accuracy_score
import numpy as np

# 使用 tqdm.notebook 展示代码运行进度
from tqdm.notebook import tqdm, trange
tqdm.pandas()

# 使用每个用户与不同 APP 的交互构建用户的历史记录文档
user_docs = train_data.groupby('uid')['appid'].progress_apply(lambda x: ''.join(map(str, x))).reset_index()

# 使用 TF-IDF 技术构建用户向量
tfidf_vectorizer = TfidfVectorizer()
user_vectors = tfidf_vectorizer.fit_transform(user_docs['appid'])

```

```

# 使用余弦相似度计算用户与每个 APP 之间的相似性
app_ids = train_data['appid'].unique()
app_vectors = tfidf_vectorizer.transform(map(str, app_ids))

similarities = cosine_similarity(user_vectors, app_vectors)

# 为每个用户推荐最相似的 APP
recommendations = np.argsort(similarities, axis=1)[:,-1]

# 评估推荐系统的性能
# 由于我们现在是在训练数据上进行推荐, 因此我们可以计算准确率
y_true = train_data.groupby('uid')['appid'].apply(lambda x: x.iloc[-1]).values # 获取每个用户最后一个交互的 APP 作为真实值
y_pred = app_ids[recommendations]

accuracy = accuracy_score(y_true, y_pred)
mae = mean_absolute_error(y_true, y_pred)

# 计算覆盖率
unique_recommended = np.unique(y_pred)
coverage = len(unique_recommended) / len(app_ids)

# 计算新颖性
overall_popularity = train_data['appid'].value_counts(normalize=True)
recommended_popularity = overall_popularity.loc[unique_recommended].mean()

# 计算流行度偏见
popularity_bias = np.sum([(overall_popularity[app] - recommended_popularity) ** 2 for app in y_pred]) / len(y_pred)

# 可视化推荐系统的评价指标
metrics = ['Accuracy', 'MAE', 'Coverage', 'Novelty', 'Popularity Bias']
values = [accuracy, mae, coverage, recommended_popularity, popularity_bias]

plt.figure(figsize=(10, 6))
plt.bar(metrics, values, color=['blue', 'green', 'red', 'cyan', 'magenta'])
plt.title('Recommendation System Evaluation Metrics')
plt.ylabel('Value')
plt.grid(axis='y')

for i, v in enumerate(values):
    plt.text(i, v + 0.01, f'{v:.4f}', ha='center', va='bottom', fontsize=9)

plt.tight_layout()
plt.show()

print(f'Accuracy: {accuracy:.4f}')
print(f'MAE: {mae:.4f}')
print(f'Coverage: {coverage:.4f}')
print(f'Novelty (Recommended Popularity): {recommended_popularity:.4f}')
print(f'Popularity Bias: {popularity_bias:.4f}')

import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import Dataset, DataLoader

```

```

from tqdm.notebook import trange, tqdm
import matplotlib.pyplot as plt

# 设置 GPU 加速
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")

# 1. 定义数据集
class RecommendationDataset(Dataset):
    def __init__(self, user_tensor, item_tensor, target_tensor):
        self.user_tensor = user_tensor
        self.item_tensor = item_tensor
        self.target_tensor = target_tensor

    def __getitem__(self, index):
        return self.user_tensor[index], self.item_tensor[index], self.target_tensor[index]

    def __len__(self):
        return self.user_tensor.size(0)

# 2. 定义 Matrix Factorization 模型
class MatrixFactorization(nn.Module):
    def __init__(self, num_users, num_items, num_factors=20):
        super(MatrixFactorization, self).__init__()
        self.user_emb = nn.Embedding(num_users, num_factors)
        self.item_emb = nn.Embedding(num_items, num_factors)

    def forward(self, user, item):
        user_embedding = self.user_emb(user)
        item_embedding = self.item_emb(item)
        return (user_embedding * item_embedding).sum(1)

# 3. 设置模型参数
num_users = train_data['uid'].nunique()
num_items = train_data['appid'].nunique()
num_factors = 20 # 设定隐因子的数量

model = MatrixFactorization(num_users, num_items, num_factors).to(device)
criterion = nn.MSELoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)

# 4. 创建数据集并训练模型
train_dataset = RecommendationDataset(
    torch.tensor(train_data['uid'].astype('category').cat.codes.values, dtype=torch.long),
    torch.tensor(train_data['appid'].astype('category').cat.codes.values, dtype=torch.long),
    torch.tensor(train_data['score'].values, dtype=torch.float32))
train_loader = DataLoader(train_dataset, batch_size=64, shuffle=True)

losses = []
for epoch in trange(5, desc='Epochs'):
    model.train()
    running_loss = 0.0
    for i, (uid, iid, label) in enumerate(tqdm(train_loader, desc='Training', leave=False)):
        uid, iid, label = uid.to(device), iid.to(device), label.to(device)

```

```

optimizer.zero_grad()
outputs = model(uid, iid)
loss = criterion(outputs, label)
loss.backward()
optimizer.step()

running_loss += loss.item()
epoch_loss = running_loss / len(train_loader)
losses.append(epoch_loss)
print(f'Epoch {epoch + 1}, Loss: {epoch_loss:.4f}')

# 5. 可视化训练过程
plt.figure(figsize=(10, 6))
plt.plot(losses, color='blue', marker='o', linestyle='dashed', linewidth=2, markersize=6)
plt.title('Training Loss Over Epochs', fontsize=16)
plt.xlabel('Epoch', fontsize=14)
plt.ylabel('MSE Loss', fontsize=14)
plt.grid(True, which='both', linestyle='--', linewidth=0.5)
plt.gca().spines['top'].set_visible(False)
plt.gca().spines['right'].set_visible(False)
plt.show()

# 6. 评估模型
test_dataset = RecommendationDataset(
    torch.tensor(test_data['uid']).astype('category').cat.codes.values, dtype=torch.long),
    torch.tensor(test_data['appid']).astype('category').cat.codes.values, dtype=torch.long),
    torch.tensor(test_data['score']).values, dtype=torch.float32))
test_loader = DataLoader(test_dataset, batch_size=64, shuffle=False)

model.eval()
test_loss = 0.0
with torch.no_grad():
    for uid, iid, label in tqdm(test_loader, desc='Testing'):
        uid, iid, label = uid.to(device), iid.to(device), label.to(device)
        outputs = model(uid, iid)
        loss = criterion(outputs, label)
        test_loss += loss.item()
    test_loss /= len(test_loader)
    print(f'Test MSE Loss: {test_loss:.4f}')

=
“=====2.2 协同过滤(Collaborative filtering)=====”

import pandas as pd
import numpy as np
from tqdm.notebook import tqdm
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.metrics import mean_absolute_error
import matplotlib.pyplot as plt

# 删除 train_data 中的重复行
train_data = train_data.drop_duplicates(subset=['uid', 'appid'])

# 1. 计算用户之间的相似度
# 使用余弦相似度计算用户之间的相似度
user_item_matrix = train_data.pivot(index='uid', columns='appid', values='score').fillna(0)
user_similarity = cosine_similarity(user_item_matrix)

```

```

user_similarity_df = pd.DataFrame(user_similarity, index=user_item_matrix.index,
                                  columns=user_item_matrix.index)

# 2. 使用相似用户的评分来预测当前用户对未评分物品的评分
def predict_ratings(user_id, app_id):
    if app_id in user_item_matrix.columns:
        similar_users = user_similarity_df[user_id].drop(user_id)
        rated_users = user_item_matrix.loc[similar_users.index, app_id].dropna()
        top_similar_users = similar_users.loc[rated_users.index]
        predicted_rating = np.dot(rated_users, top_similar_users) / top_similar_users.sum()
        return predicted_rating
    else:
        return 3 # 默认评分

# 为测试集预测评分
tqdm.pandas()
test_data['predicted_score'] = test_data.progress_apply(lambda x: predict_ratings(x['uid'], x['appid']),
                                                         axis=1)

# 3. 基于预测评分为用户推荐物品
top_n_recommendations = test_data.groupby('uid').apply(lambda x: x.nlargest(5,
                                     'predicted_score')['appid'].tolist())

# 4. 使用 Accuracy、MAE、Coverage、Novelty、Popularity Bias 等指标评估模型
# Accuracy
accuracy = (test_data['score'] == test_data['predicted_score']).mean()

# MAE
mae = mean_absolute_error(test_data['score'], test_data['predicted_score'])

# Coverage
coverage = len(top_n_recommendations.explode().unique()) / len(user_item_matrix.columns)

# Novelty
item_popularity = train_data['appid'].value_counts(normalize=True)
novelty = top_n_recommendations.explode().map(item_popularity).mean()

# Popularity Bias
popularity_bias = top_n_recommendations.explode().map(item_popularity).std()

print(f"Accuracy: {accuracy:.4f}")
print(f"MAE: {mae:.4f}")
print(f"Coverage: {coverage:.4f}")
print(f"Novelty: {novelty:.4f}")
print(f"Popularity Bias: {popularity_bias:.4f}")

# 5. 可视化训练过程
plt.figure(figsize=(10, 6))
plt.plot(test_data['score'], label='True Ratings', alpha=0.7)
plt.plot(test_data['predicted_score'], label='Predicted Ratings', alpha=0.7)
plt.title('True vs. Predicted Ratings')
plt.xlabel('Test Data Index')
plt.ylabel('Rating')
plt.legend()
plt.show()

```

“=====2.3 矩阵分解(Matrix Factorization)=====”

```
import numpy as np
import pandas as pd
from sklearn.metrics import mean_absolute_error
from scipy.sparse import csr_matrix
from scipy.sparse.linalg import svds
from tqdm.notebook import tqdm
import matplotlib.pyplot as plt

# 将数据转为用户-APP 评分矩阵
user_app_matrix = train_data.pivot(index='uid', columns='appid', values='score')
user_app_matrix.fillna(0, inplace=True)

# 用户和 APP 的 ID 到索引的映射
user_to_idx = {uid: i for i, uid in enumerate(user_app_matrix.index)}
app_to_idx = {appid: j for j, appid in enumerate(user_app_matrix.columns)}
idx_to_user = {i: uid for uid, i in user_to_idx.items()}
idx_to_app = {j: appid for appid, j in app_to_idx.items()}

# 将 DataFrame 转为 numpy 数组
user_app_matrix_np = user_app_matrix.values

# 对用户评分进行中心化处理（减去每个用户的平均评分）
user_ratings_mean = np.mean(user_app_matrix_np, axis=1)
user_app_demeaned = user_app_matrix_np - user_ratings_mean.reshape(-1, 1)

# 使用 SVD 进行矩阵分解
U, sigma, Vt = svds(user_app_demeaned, k=50)
sigma = np.diag(sigma)

# 使用 U, Sigma, Vt 得到预测评分矩阵
predicted_ratings = np.dot(np.dot(U, sigma), Vt) + user_ratings_mean.reshape(-1, 1)
predicted_ratings_df = pd.DataFrame(predicted_ratings, columns=user_app_matrix.columns,
index=user_app_matrix.index)

# 对测试集进行预测
test_uids = test_data['uid'].values
test_appids = test_data['appid'].values
actual_scores = test_data['score'].values
predicted_scores = []

for uid, appid in tqdm(zip(test_uids, test_appids), total=len(test_uids)):
    if uid in user_to_idx and appid in app_to_idx:
        predicted_scores.append(predicted_ratings_df.loc[uid, appid])
    else:
        predicted_scores.append(np.nan)

# 评估指标

# 1. Accuracy
predicted_binary = [1 if score >= 3 else 0 for score in predicted_scores]
actual_binary = [1 if score >= 3 else 0 for score in actual_scores]
accuracy = np.mean(np.array(predicted_binary) == np.array(actual_binary))

# 2. MAE
mae = mean_absolute_error(actual_scores, predicted_scores)
```

```
print(f'Accuracy: {accuracy} ")
print(f'Mean Absolute Error: {mae} ")
```

```
# 计算覆盖率
```

```
def coverage(predicted_ratings, threshold=3):
    """
    计算推荐系统的覆盖率
    Args:
    - predicted_ratings: 预测的评分矩阵
    - threshold: 推荐的阈值, 当评分超过此值时认为是推荐的

    Returns:
    - 覆盖率
    """
    num_recommended = np.sum(predicted_ratings > threshold)
    total_items = np.prod(predicted_ratings.shape)
    return num_recommended / total_items
```

```
# 计算新颖性
```

```
def novelty(predicted_ratings_df, train_data):
    """
    计算推荐系统的新颖性
    Args:
    - predicted_ratings_df: 预测的评分 DataFrame
    - train_data: 训练数据

    Returns:
    - 新颖性
    """
    # 计算每个项目的流行度
    item_popularity = train_data.groupby('appid').size().reset_index(name='popularity')
    item_popularity_dict = dict(zip(item_popularity['appid'], item_popularity['popularity']))

    # 计算推荐项目的平均流行度
    recommended_items = predicted_ratings_df.stack().reset_index()
    recommended_items.columns = ['uid', 'appid', 'score']
    recommended_items['popularity'] = recommended_items['appid'].apply(lambda x:
item_popularity_dict.get(x, 0))

    return recommended_items['popularity'].mean()
```

```
# 计算流行度偏见
```

```
def popularity_bias(predicted_ratings_df, train_data):
    """
    计算推荐系统的流行度偏见
    Args:
    - predicted_ratings_df: 预测的评分 DataFrame
    - train_data: 训练数据

    Returns:
    - 流行度偏见
```

```

"""
# 计算每个项目的流行度
item_popularity = train_data.groupby('appid').size().reset_index(name='popularity')
item_popularity_dict = dict(zip(item_popularity['appid'], item_popularity['popularity']))

# 计算推荐项目的平均流行度
recommended_items = predicted_ratings_df.stack().reset_index()
recommended_items.columns = ['uid', 'appid', 'score']
recommended_items['popularity'] = recommended_items['appid'].apply(lambda x:
item_popularity_dict.get(x, 0))

# 计算流行项目和非流行项目的推荐比例
total_recommended = len(recommended_items)
popular_recommended = np.sum(recommended_items['popularity'] >
recommended_items['popularity'].median())

return popular_recommended / total_recommended

# 计算上述评估指标
cov = coverage(predicted_ratings)
nov = novelty(predicted_ratings_df, train_data)
pop_bias = popularity_bias(predicted_ratings_df, train_data)

print(f"Coverage: {cov}")
print(f"Novelty: {nov}")
print(f"Popularity Bias: {pop_bias}")

# 可视化训练过程
plt.plot(sigma)
plt.title("Singular Values")
plt.xlabel("Component")
plt.ylabel("Value")
plt.show()

```

“=====2.4 深度学习方法（NCF）=====”

```

import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import Dataset, DataLoader
from tqdm.notebook import trange, tqdm
import matplotlib.pyplot as plt
import numpy as np

# 设置 GPU 加速
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")

# 1. 定义数据集
class NCFDataset(Dataset):
    def __init__(self, user_tensor, item_tensor, target_tensor):
        self.user_tensor = user_tensor
        self.item_tensor = item_tensor
        self.target_tensor = target_tensor

```

```

def __getitem__(self, index):
    return self.user_tensor[index], self.item_tensor[index], self.target_tensor[index]

def len(self):
    return self.user_tensor.size(0)

```

2. 定义 NCF 模型

```

class NCF(nn.Module):
    def __init__(self, num_users, num_items, latent_dim):
        super(NCF, self).__init__()
        self.user_emb = nn.Embedding(num_users, latent_dim)
        self.item_emb = nn.Embedding(num_items, latent_dim)
        self.fc_layers = nn.Sequential(
            nn.Linear(2 * latent_dim, 128),
            nn.ReLU(),
            nn.Linear(128, 64),
            nn.ReLU(),
            nn.Linear(64, 1)
        )

    def forward(self, user, item):
        user_embedding = self.user_emb(user)
        item_embedding = self.item_emb(item)
        x = torch.cat([user_embedding, item_embedding], dim=1)
        x = self.fc_layers(x)
        return x.squeeze()

```

```

num_users = train_data['uid'].nunique()
num_items = train_data['appid'].nunique()
latent_dim = 50

```

```

model = NCF(num_users, num_items, latent_dim).to(device)
criterion = nn.MSELoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)

```

为 uid 和 appid 列分配唯一的整数 ID

```

train_data['uid'] = train_data['uid'].factorize()[0]
train_data['appid'] = train_data['appid'].factorize()[0]
test_data['uid'] = test_data['uid'].factorize()[0]
test_data['appid'] = test_data['appid'].factorize()[0]

```

创建数据集并训练模型

```

train_dataset = NCFDataset(torch.tensor(train_data['uid'].values, dtype=torch.long),
                             torch.tensor(train_data['appid'].values, dtype=torch.long),
                             torch.tensor(train_data['score'].values, dtype=torch.float32))
train_loader = DataLoader(train_dataset, batch_size=64, shuffle=True)

```

```

losses = []

```

```

for epoch in trange(5, desc='Epochs'):
    model.train()
    running_loss = 0.0
    for i, (uid, iid, label) in enumerate(tqdm(train_loader, desc='Training', leave=False)):
        uid, iid, label = uid.to(device), iid.to(device), label.to(device)

        optimizer.zero_grad()

```

```

        outputs = model(uid, iid)
        loss = criterion(outputs, label)
        loss.backward()
        optimizer.step()

        running_loss += loss.item()
        epoch_loss = running_loss / len(train_loader)
        losses.append(epoch_loss)
        print(f'Epoch {epoch + 1}, Loss: {epoch_loss:.4f}')

# 4. 可视化训练过程
plt.plot(losses)
plt.title('Training Loss Over Epochs')
plt.xlabel('Epoch')
plt.ylabel('MSE Loss')
plt.grid(True)
plt.show()

# 5. 评估模型
test_dataset = NCFDataset(torch.tensor(test_data['uid'].values, dtype=torch.long),
                           torch.tensor(test_data['appid'].values, dtype=torch.long),
                           torch.tensor(test_data['score'].values, dtype=torch.float32))
test_loader = DataLoader(test_dataset, batch_size=64, shuffle=False)

model.eval()
predicted_scores = []
true_scores = []
all_items = set(test_data['appid'].unique())
recommended_items = set()

with torch.no_grad():
    test_loss = 0.0
    mae_loss = 0.0
    for uid, iid, label in tqdm(test_loader, desc='Testing'):
        uid, iid, label = uid.to(device), iid.to(device), label.to(device)
        outputs = model(uid, iid)

        # Calculate losses
        mse_loss = criterion(outputs, label)
        test_loss += mse_loss.item()
        mae_loss += torch.mean(torch.abs(outputs - label)).item()

        # For accuracy, coverage, and novelty
        predicted_scores.extend(outputs.cpu().numpy())
        true_scores.extend(label.cpu().numpy())
        recommended_items.update(iid.cpu().numpy())

test_loss /= len(test_loader)
mae_loss /= len(test_loader)

# Accuracy: Assuming a threshold of 0.5 for liking an item
predictions = [1 if score > 0.5 else 0 for score in predicted_scores]
true_labels = [1 if score > 0.5 else 0 for score in true_scores]
accuracy = sum([p == t for p, t in zip(predictions, true_labels)]) / len(predictions)

# Coverage
coverage = len(recommended_items) / len(all_items)

```

```
# Popularity Bias (assuming item popularity is the average score of an item)
item_popularity = test_data.groupby('appid')['score'].mean().to_dict()
avg_recommended_popularity = np.mean([item_popularity.get(item, 0) for item in
recommended_items])
avg_all_popularity = np.mean(list(item_popularity.values()))

# Novelty
all_items_ranked_by_popularity = [item[0] for item in
sorted(item_popularity.items(), key=lambda x: x[1],
reverse=True)]
avg_rank_recommended = np.mean([all_items_ranked_by_popularity.index(item) for item in
recommended_items])
novelty = 1 - avg_rank_recommended / len(all_items_ranked_by_popularity)

print(f"Test MSE Loss: {test_loss:.4f}")
print(f"Test MAE Loss: {mae_loss:.4f}")
print(f"Accuracy: {accuracy:.4f}")
print(f"Coverage: {coverage:.4f}")
print(f"Novelty: {novelty:.4f}")
print(f"Popularity Bias (Recommended/All):
{avg_recommended_popularity:.4f}/{avg_all_popularity:.4f}")
```

周铠然 · 仅供求职展示