**餐饮大数据智能推荐策略**

小组成员：陈龙泉，莫镇玮，黄志杰，冯梓豪

成员分工：

工作前期：在本阶段，小组成员聚集起来共同讨论项目，研究项目背景，明确项目目标，分析项目需求，确定解决思路，并根据每个人的特点进行分工。

工作中期：

在本阶段，由陈龙泉负责项目代码的主体实现。其余组员负责提供项目改进意见以及协助解决可能的代码错误

工作后期：

在本阶段，由莫镇玮负责制作汇报PPT，并由冯梓豪负责进行项目成果汇报。其余组员负责补充可能遗漏的内容以及提供改进建议。

工作尾期：

在本阶段，由黄志杰负责汇总项目内容和成果并攥写word文档。其余组员负责检查攥写的word文档是否有缺漏和错误。

# 项目介绍

广大上班族已经习惯于在餐饮外卖平台上订餐。W餐饮外卖平台，向广大用户提供网上订餐服务。W平台订餐完成后，平台会引导用户对于品尝过的菜品进行评价打分。运营方发现老用户的下单率呈现下降态势，市场部门建议，希望针对老用户进行个性化的菜品推荐，包括用户的偏爱菜品及新菜品。

W餐饮平台的技术部门决定根据近期用户对菜品的评分历史数据，建立菜品推荐模型，向用户们提供菜品推荐。菜品智能推荐系统，作为原来的餐饮外卖平台系统的扩展与补充，主要负责对用户的历史评分数据进行处理，并生成推荐结果集。

本文提供了平台2个月的用户评分数据及菜品数据，通过数据分析，为平台制定综合的菜品推荐策略，以便更好地服务线上用户。

## 餐饮大数据智能推荐背景与目标

随着互联网、移动互联网和各种智能设备的普及，人们在日常生活中产生了大量的数据。用户在订餐、点评、社交媒体上的行为、消费偏好等信息被记录下来，形成了庞大的餐饮相关数据。传统的餐饮推荐方法如口碑、推荐网站等无法满足用户个性化的需求。人们希望根据自己的喜好、地理位置、健康需求等因素，得到更准确、个性化的餐饮推荐，提高用餐体验。

随着大数据时代的到来，数据分析技术不断发展和成熟，数据分析技术不断进步，机器学习、数据挖掘、自然语言处理等技术的进步使得对大规模餐饮数据的处理和分析变得更加高效和精确。餐饮行业竞争激烈，餐厅和美食推广需要更好的方式和工具。通过分析大数据，餐厅可以了解消费者的需求、喜好和趋势，从而精确推送符合用户口味的推荐，增加销售额和用户黏性。

基于以上背景，餐饮大数据智能推荐技术应运而生。它通过收集、整理和分析庞大的餐饮数据，使用机器学习和数据挖掘等算法，为用户提供个性化的餐饮推荐，同时也为餐饮企业提供精准的市场分析和决策支持。这一技术的发展和应用有助于优化用户体验、促进餐饮行业的发展，并提升企业竞争力。

本次餐饮大数据智能推荐目标如下：

根据平台2个月的用户评分数据及菜品数据，进行数据分析，为平台制定综合的菜品推荐策略，以便更好地服务线上用户。

## 常见的餐饮大数据智能推荐

常见的餐饮大数据智能推荐算法有：

①协同过滤法：基于用户行为和偏好，通过分析用户与其他相似用户的共同兴趣来进行推荐。例如，根据用户的历史评分数据，找到相似口味偏好的用户，并向其推荐相似喜欢的餐厅或菜品。

②内容过滤法：基于餐厅或菜品的属性信息，通过分析用户的历史偏好以及餐厅或菜品的特征，为用户推荐具有相似属性的餐厅或菜品。例如，根据用户的喜好和口味偏好，向其推荐相似类型的餐厅或菜品。

③基于关联规则的推荐法：通过挖掘用户历史购买记录中的频繁项集和关联规则，为用户推荐潜在的相关餐厅或菜品。例如，如果用户经常同时购买汉堡和可乐，可以向用户推荐购买汉堡时附带的可乐。

④基于深度学习的推荐法：利用深度神经网络模型，对用户和餐厅（或菜品）进行表示学习，通过训练模型来预测用户对餐厅或菜品的偏好，并为其推荐相关的餐厅或菜品

## 餐饮大数据智能推荐步骤与思路

本次餐饮推荐主要包括以下步骤：

①对数据进行探索：部分数据的属性存在缺失、重复、异常的情况，例如属性的值为NULL，对于这些数据，我们在预处理时予以剔除。

②预处理：根据需要对数据集进行统计，如每个用户的评论次数，每个用户对每个菜品的评论次数。划分数据集，80%的数据被划分为训练集，20%的数据被划分为测试集。

③在第上一步得到的数据基础上，构建建模样本，训练模型：设置ALS模型参数并拟合ALS模型。使用测试集进行预测，并通过计算计算均方误差RMSE来评估模型性能。

④展示推荐策略的结果：

为所有用户推荐前10个菜品；

为所有菜品推荐前10个用户；

为部分用户推荐前10个菜品；

为部分菜品推荐前10个用户。

## 2.1 数据说明与存储

1. **用户评分数据集**

|  |  |
| --- | --- |
| **字段名** | **描述** |
| UserID | 用户ID |
| Rating | 评分 |
| ReviewTime | 评分的时间戳 |
| Review | 评论 |
| MealID | 菜品ID |

UserID: 用户ID，用于唯一标识每个用户。

Rating: 用户对菜品的评分，通常是一个数值，表示用户对菜品的满意程度或喜好程度。

ReviewTime: 评分的时间戳，记录用户进行评分的具体时间。

Review: 用户对菜品的评论文本，可以是用户对菜品口味、服务等方面的评价。

MealID: 菜品ID，与菜品数据集中的菜品ID对应，用于将用户评分与具体的菜品关联起来。

1. **菜品数据集**

|  |  |
| --- | --- |
| **字段名** | **描述** |
| mealno | 序号 |
| mealID | 菜品ID |
| meal\_name | 菜品名称 |

菜品数据集包含了餐厅提供的菜品信息。其中，字段说明如下：

mealno: 序号，用于唯一标识每个菜品。

mealID: 菜品ID，与用户评分数据集中的MealID对应，用于将菜品与用户评分数据关联起来。

meal\_name: 菜品名称，描述了菜品的名字或品类。

这两个数据集可以结合使用，通过用户评分数据集中的UserID和MealID字段，可以找到特定用户对特定菜品的评分信息，并根据菜品数据集中的meal\_name字段了解菜品的名称和类别。这些信息可以用于基于用户偏好的推荐算法、菜品排名和分析用户对不同菜品的反馈等应用场景。

## 2.2.1 数据分析展示

代码段

print("菜品评论数据集的行数:")

println(MealRatingsdata.count()) // 输出数据集的行数

print("菜品评论数据集的结构:")

MealRatingsdata.printSchema() // 打印数据集的结构

运行结果

println("显示原始数据集:")

MealRatingsdata.show(10)

## 2.2.2 处理缺失值

代码段

// 剔除任何一个属性有缺失值的行

val MealRatingslostData = MealRatingsdata.na.drop(Seq("UserID", "Rating", "ReviewTime", "Review", "MealID"))

// 计算处理后的数据行数

val lostrowCount = MealRatingslostData.count()

//缺失属性的行数

val lostnum=MealRatingsdata.count()-lostrowCount

println(s"处理的缺失行数: $lostnum")

println(s"处理后的数据行数: $lostrowCount")

// 打印处理后的数据集

println(s"处理后的数据集:")

MealRatingslostData.show(10)

运行结果

首先，na.drop(Seq("UserID", "Rating", "ReviewTime", "Review", "MealID"))函数用于剔除具有缺失值的行。在此代码中，我们使用了Seq来指定需要检查缺失值的属性列，即"UserID"、"Rating"、"ReviewTime"、"Review"和"MealID"。这将返回一个新的数据集MealRatingslostData，其中已经删除了包含缺失值的行。

接下来，通过计算lostrowCount = MealRatingslostData.count()，获取处理后的数据行数，即剔除了含有缺失值的行后的数据集的行数。

然后，通过lostnum=MealRatingsdata.count()-lostrowCount，计算出原始数据集中缺失属性的行数。这里使用原始数据集MealRatingsdata减去处理后的数据行数lostrowCount即可得到仅包含缺失值的行数。

最后，通过println语句将处理后的缺失行数lostnum和处理后的数据行数lostrowCount输出。同时，使用show(10)打印处理后的数据集的前10行，以便查看处理后的结果。处理缺失值的目的是确保数据的完整性和准确性。通过剔除含有缺失值的行，可以避免在后续的数据分析和建模过程中由于缺失值引起的错误或偏差。

## 2.2.3 处理重复值

代码段

// 去除重复的行

val processedrepeData = FinaMealRatingsData.dropDuplicates(Seq("UserID", "MealID"))

val ReperowCount = processedrepeData.count()

//缺失属性的行数

val repenum=MealRatingsdata.count()-ReperowCount

println(s"处理的重复行数: $repenum")

println(s"处理的重复数据集：")

processedrepeData.show()

运行结果

通过dropDuplicates(Seq("UserID", "MealID"))函数对数据集进行去重操作。在这里，我们使用了Seq来指定需要检查重复行的属性列，即"UserID"和"MealID"。这将返回一个新的数据集processedrepeData，其中已经去除了重复的行。

接下来，通过ReperowCount = processedrepeData.count()获取去重后的数据行数，即去除重复行后的数据集的行数。

然后，通过repenum=MealRatingsdata.count()-ReperowCount计算出原始数据集中重复属性的行数。这里使用原始数据集MealRatingsdata减去去重后的数据行数ReperowCount即可得到仅包含重复值的行数。

去除重复行的目的是确保数据的唯一性和准确性。重复的行可能会引入噪音或导致结果的偏倚，因此在数据分析和建模过程中需要将其剔除。去重操作有助于提高数据的质量和准确性，并减少重复样本在分析中的影响。

**处理后的数据集：**

println(s"处理后的数据行数: $ReperowCount")

// 打印处理后的数据集

println(s"处理后的数据集: ")

FinaMealRatingsData.show()//显示最终的菜品评论数据集

## 2.2.4 STRING类型属性ID转码

代码段

// 创建StringIndexer并转换MealID列

val MealIdIndexer= new StringIndexer().setInputCol("MealID").setOutputCol("MealIDIndex")

val indexedData1 = MealIdIndexer.fit(FinaMealRatingsData).transform(FinaMealRatingsData)

// 创建StringIndexer并转换UserID列

val userIdIndexer = new StringIndexer().setInputCol("UserID").setOutputCol("UserIDIndex")

val indexedData2 = userIdIndexer.fit(indexedData1).transform(indexedData1)

//并保存到hive表中

indexedData2.write.mode("overwrite").format("parquet").saveAsTable("rating\_data\_table")

println("转码后的数据集：")

indexedData2.show()

运行结果

首先，我们创建了一个StringIndexer对象MealIdIndexer，并设置输入列为"MealID"，输出列为"MealIDIndex"。接着，使用fit方法在数据集FinaMealRatingsData上进行拟合，并将拟合结果应用于数据集，生成一个新的数据集indexedData1。这个新的数据集中，"MealID"列被转换为数值类型的"MealIDIndex"列。

接下来，我们创建了另一个StringIndexer对象userIdIndexer，并设置输入列为"UserID"，输出列为"UserIDIndex"。同样地，使用fit方法在前一步骤生成的数据集indexedData1上进行拟合，并将拟合结果应用于数据集，生成最终的转码数据集indexedData2。在indexedData2中，"UserID"列被转换为数值类型的"UserIDIndex"列。

最后，通过write方法将转码后的数据集indexedData2保存到Hive表中。使用"overwrite"模式表示如果表已存在，则覆盖原有表。文件格式选择为Parquet格式，并指定表名为"rating\_data\_table"。

然后写进Hive表中

// 查询并显示写入Hive表后的内容

val ratingtableData= spark.sql("SELECT \* FROM rating\_data\_table")

println("写入Hive表后的内容：")

ratingtableData.show()

## 2.2.5 读取菜品数据集

代码段

//读取MySQL数据库中的meal\_list表格内容

val MealListData: DataFrame = spark.read

.format("jdbc") //指定数据源格式为JDBC

.option("url", "jdbc:mysql://master:3306") //指定数据库连接URL

.option("driver", "com.mysql.cj.jdbc.Driver") //指定JDBC驱动程序类

.option("dbtable", "data.meal\_list") //指定要读取的数据表名称

.option("user", "root") //指定数据库用户名

.option("password", "123456") //指定数据库密码

.load() //加载数据

print("菜品名数据集的行数:")

println(MealListData.count()) // 输出数据集的行数

print("菜品名数据集的结构:")

MealListData.printSchema() // 打印数据集的结构

MealListData.show(10)//默认只显示20行

运行结果

## 2.2.6 异常数据处理

代码段

//挑选有异常的数据行

val randData = MealListData.filter(col("mealno").isNull || col("mealID").isNull || col("meal\_name").isNull)

// 删除具有任意属性为NULL值的行

val finaMealListData = MealListData

.filter(col("mealno").isNotNull && col("mealID").isNotNull && col("meal\_name").isNotNull)

// 统计处理后的数据行数

val rowCount = finaMealListData.count()

val randnum=MealListData.count()-rowCount

println(s"处理的数据数: $randnum")

println(s"处理的数据集:")

randData.show()

// 显示处理后的数据行数

println(s"处理后的数据数: $rowCount")

println(s"处理后的数据集:")

finaMealListData.show()//显示最终的菜品名单数据集

运行结果

这里使用filter函数挑选出具有异常值的数据行，条件是其中的mealno、mealID或meal\_name至少一个为NULL，将挑选出来的异常数据行存储在randData变量中。然后，使用filter函数删除具有任意属性为NULL值的数据行，筛选出没有异常值的数据行，将处理后的数据行存储在finaMealListData变量中。通过使用count函数统计处理后的数据行数，将结果分别存储在rowCount和randnum变量中。最后，使用println函数分别打印出处理的数据数和处理后的数据数，并使用show函数显示异常数据集randData和最终的菜品名单数据集finaMealListData。

## 2.2.7用户评论次数和占比

代码段

// 计算每个用户的评论次数，并统计每个用户评论的次数和占比。

val userCount = FinaMealRatingsData.groupBy("UserID").count()

val totalCount = userCount.agg(sum("count")).first().getLong(0)

val userStats = userCount.withColumn("percentage", col("count") / totalCount \* 100)

// 打印每个用户评论的次数和占比

println("每个用户的评论次数和占比：")

userStats.show()

运行结果

这段代码用于对数据集FinaMealRatingsData进行处理，计算每个用户的评论次数，并统计每个用户评论次数的占比。具体实现过程为：首先使用groupBy函数按照"UserID"列对数据集进行分组，并使用count函数计算每个用户的评论次数，然后使用agg函数和sum函数对评论次数进行聚合操作，得到所有用户评论次数的总和。接着添加一个新列"percentage"，该列的值为每个用户评论次数与总和的比例乘以100，即每个用户评论次数的占比。最后，使用println函数打印出"每个用户的评论次数和占比："，并使用show函数显示每个用户评论次数和占比的数据集userStats。

## 2.2.8模型预测

代码段

// 将数据集划分为训练集和测试集。

val Array(trainingData, testData) = ALSData.randomSplit(Array(0.8, 0.2))

// 将训练集创建并保存到新表MealRatingsDataTrain表中

trainingData.write.mode(SaveMode.Overwrite).saveAsTable("MealRatingsDataTrain")

// 设置ALS模型参数。

val als = new ALS()

.setMaxIter(10)

.setRegParam(0.01)

.setUserCol("UserIDIndex")

.setItemCol("MealIDIndex")

.setRatingCol("Rating")

// 拟合ALS模型。

val model = als.fit(trainingData)

//使用测试集进行预测，并保存结果到DataFrame。

val predictions = model.transform(testData)

运行结果：

// 评估模型性能，计算均方误差RMSE。

val evaluator = new RegressionEvaluator()

.setMetricName("rmse")

.setLabelCol("Rating")

.setPredictionCol("prediction")

val rmse = evaluator.evaluate(predictions)

println(s"均方根误差（RMSE）: $rmse")

// 11.输出模型的用户因子矩阵和菜品因子矩阵。

val userFactors = model.userFactors

val mealFactors = model.itemFactors

println("用户因子矩阵：")

userFactors.show()

println("菜品因子矩阵：")

mealFactors.show()

运行结果：

这段代码实现了基于ALS（Alternating Least Squares）算法的推荐系统模型训练和评估过程

将原始数据集划分为训练集和测试集，划分比例为80%训练集和20%测试集，然后将训练集数据保存到新的表"MealRatingsDataTrain"中，用于后续的模型训练。设置ALS模型的参数，包括迭代次数（setMaxIter）、正则化参数（setRegParam）、用户列名（setUserCol）、菜品列名（setItemCol）和评分列名（setRatingCol）等。接着使用训练集数据对ALS模型进行训练，得到模型对象。使用测试集数据对训练好的模型进行预测，生成预测结果DataFrame。

使用回归评估器（RegressionEvaluator）计算预测结果的均方根误差（RMSE），用于评估模型的性能，最后获取训练好的模型中的用户因子矩阵（userFactors）和菜品因子矩阵（mealFactors）。

该段代码通过划分训练集和测试集，使用ALS算法进行模型训练，评估模型性能并输出均方根误差（RMSE），最后输出训练好的模型中的用户因子矩阵和菜品因子矩阵，用于推荐系统的预测和推荐结果生成。

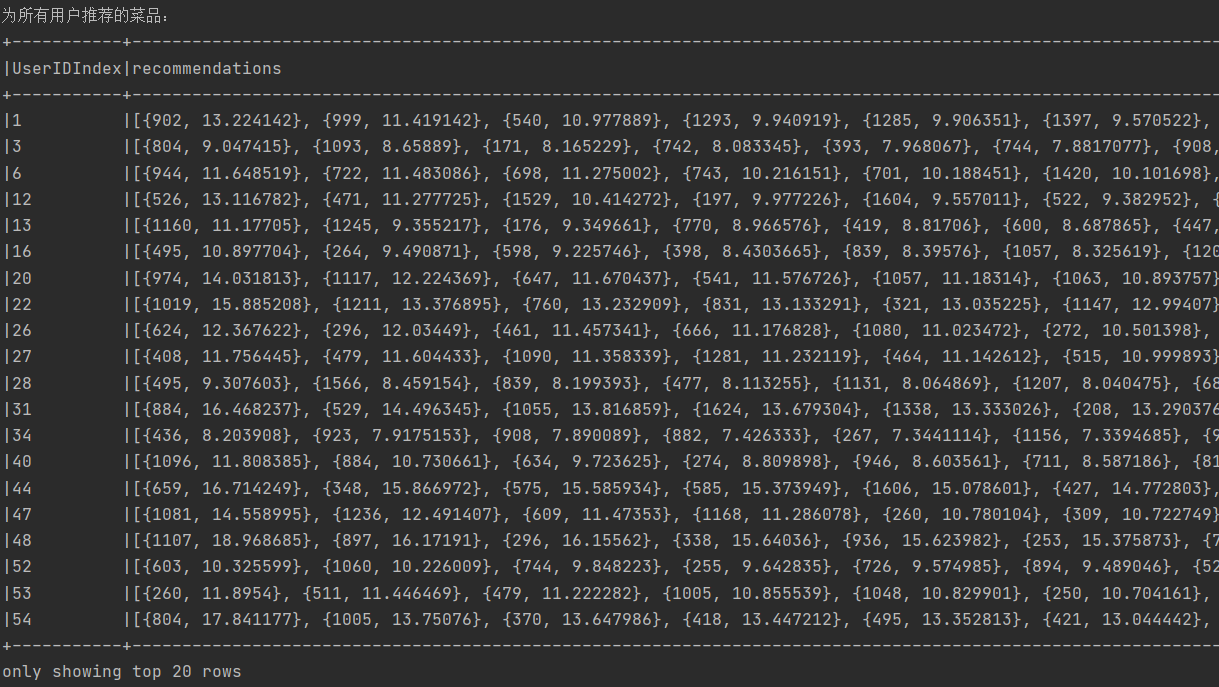
## 推荐展示

**3.1 所有用户推荐前十菜品**

// 12.为所有用户推荐前10个菜品，并输出结果。

val allUsers = ALSData.select("UserIDIndex").distinct()

val allUserRecommendations = model.recommendForUserSubset(allUsers, 10)



**3.2 连接查询得到结果**

// 将 expandedRecommendations 表与 ratingMealData 表连接

val joinedResult1= expandedRecommendations1.join(ratingMealData, Seq("MealIDIndex")).dropDuplicates()



**3.3 为菜品推荐用户**

// 展开 recommendations 列

val expandedRecommendations2= allUserRecommendations.select(

col("UserIDIndex"),

explode(col("recommendations")).as("recommendation")

).select(

col("UserIDIndex"),

col("recommendation.MealIDIndex").alias("MealIDIndex"),

col("recommendation.rating")

// 将 expandedRecommendations 表与 ratingMealData 表连接

val joinedResult2= expandedRecommendations2.join(ratingMealData, Seq("MealIDIndex")).dropDuplicates()

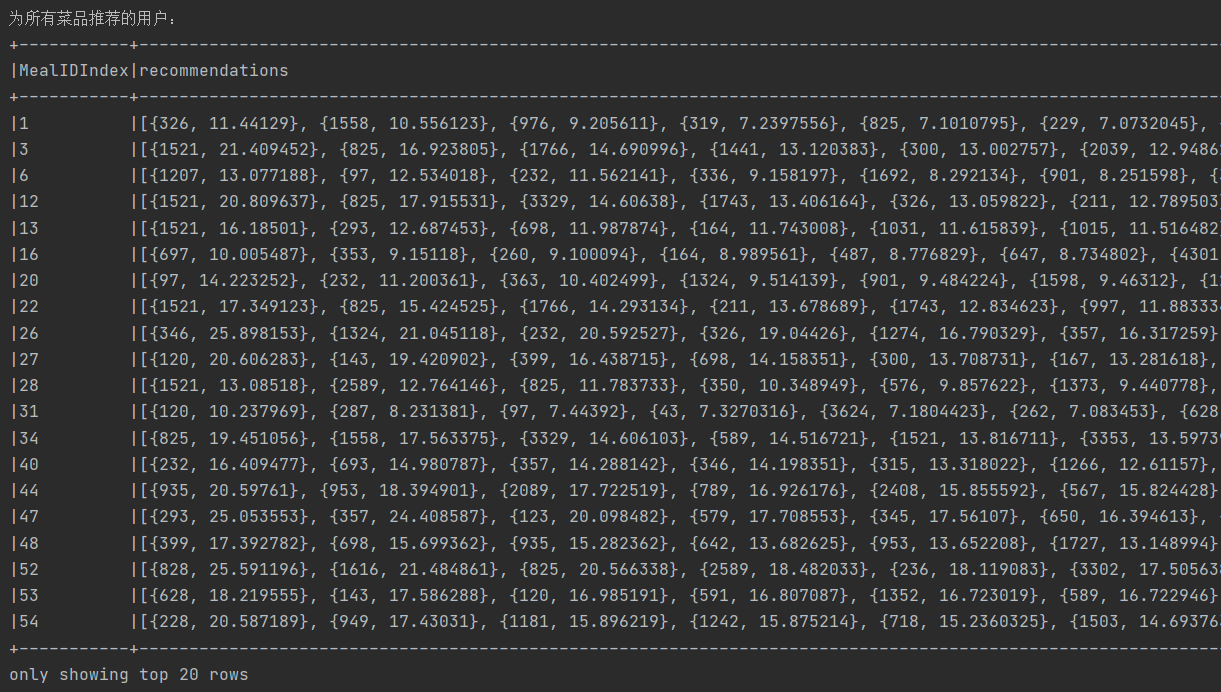
.dropDuplicates()

// 打印连接后的结果

println("为所有菜品推荐的用户及对应的菜名：")

joinedResult2.select("UserIDIndex", "MealIDIndex", "meal\_name").show(false)





首先，使用explode函数将recommendations列展开为单独的行，其中每一行包含一个推荐结果。通过select函数选择需要的列，包括UserIDIndex、MealIDIndex和rating。

接下来，使用join函数将展开后的expandedRecommendations2表与评分数据表ratingMealData进行连接，连接条件是MealIDIndex列。为了去除重复的行，使用dropDuplicates()函数进行去重。

最后，打印连接后的结果。使用select函数选择需要的列，包括UserIDIndex、MealIDIndex和meal\_name（菜名）。使用show(false)函数显示结果，其中false参数表示不截断列内容。

**3.4 为部分用户推荐菜品**

// 14.为部分用户推荐前10个菜品，并输出结果。

val someUsers = ALSData.select("UserIDIndex").distinct().limit(5)

val someUserRecommendations = model.recommendForUserSubset(someUsers, 10)

println("为部分用户推荐的菜品：")

someUserRecommendations.show(false)

val expandedRecommendations3= allUserRecommendations.select(

col("UserIDIndex"),

explode(col("recommendations")).as("recommendation")

).select(

col("UserIDIndex"),

col("recommendation.MealIDIndex").alias("MealIDIndex"),

col("recommendation.rating")

)

// 将 expandedRecommendations 表与 ratingMealData 表连接

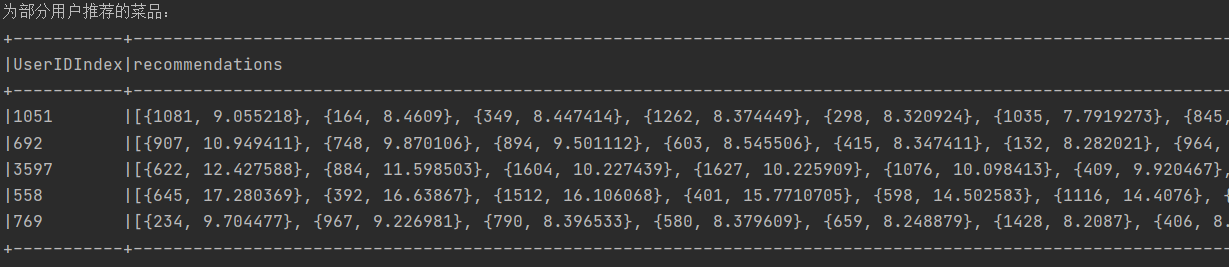
val joinedResult3= expandedRecommendations3.join(ratingMealData, Seq("MealIDIndex")).dropDuplicates()

// 打印连接后的结果

println("为部分用户推荐的菜品：")

joinedResult3.select("UserIDIndex", "MealIDIndex", "meal\_name").show(false)





这段代码实现了对一部分用户进行菜品推荐，并输出了推荐结果。首先，从数据集中选择了5个不重复的用户，并使用模型对这些用户进行推荐，每个用户推荐10个菜品。然后，将推荐结果与评分数据进行连接，得到了每个用户的推荐菜品及其相关信息。最后，将部分用户的推荐菜品及其相关信息进行输出展示。通过这段代码，可以方便地查看部分用户的推荐结果，以帮助用户决定他们可能感兴趣的菜品。

**3.4 为部分菜品推荐用户**

// 15.为部分菜品推荐前10个用户，并输出结果。

val someMeals = ALSData.select("MealIDIndex").distinct().limit(5)

val someMealRecommendations = model.recommendForItemSubset(someMeals, 10)

println("为部分菜品推荐的用户：")

someMealRecommendations.show(false)

val expandedRecommendations4= allUserRecommendations.select(

col("UserIDIndex"),

explode(col("recommendations")).as("recommendation")

).select(

col("UserIDIndex"),

col("recommendation.MealIDIndex").alias("MealIDIndex"),

col("recommendation.rating")

)

// 将 expandedRecommendations 表与 ratingMealData 表连接

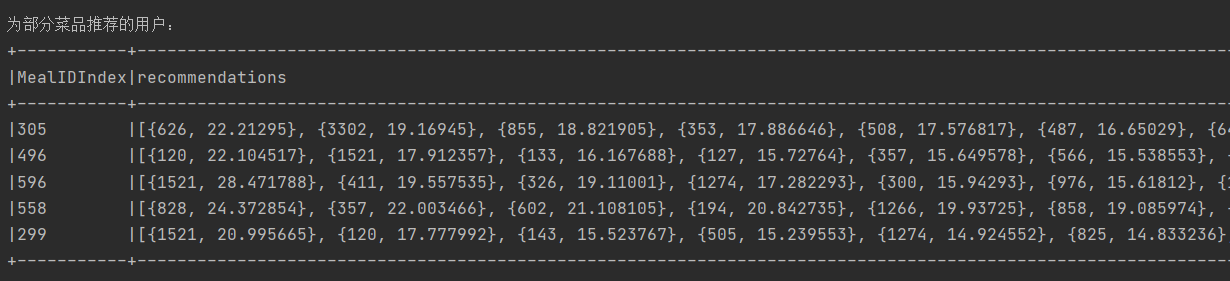
val joinedResult4= expandedRecommendations4.join(ratingMealData, Seq("MealIDIndex")).dropDuplicates()

// 打印连接后的结果

println("为部分菜品推荐的用户：")

joinedResult4.select("UserIDIndex", "MealIDIndex", "meal\_name").show(false)





这段代码实现了基于ALS模型的菜品推荐功能，其将被推荐的部分菜品作为输入，然后预测出对于每个菜品，最适合它的前10个用户。接着，代码会将这些推荐结果与原始的评分数据进行比较，以获取有关推荐的额外信息。最后，输出结果包括部分菜品和它们的推荐用户及组成菜品的食材。通过这样的推荐功能，用户可以更容易地发现他们可能感兴趣的菜品，并且能够获得关于菜品的进一步信息，以便更好地选择下一步操作。

## 4.0完整详细代码

package test

import org.apache.spark.ml.recommendation.ALS

import org.apache.spark.sql.functions.\_

import org.apache.spark.sql.expressions.Window

import org.apache.spark.ml.evaluation.RegressionEvaluator

import org.apache.spark.sql.functions.{count, desc, format\_number, sum}

import org.apache.spark.sql.{DataFrame, SaveMode, SparkSession}

import org.apache.spark.ml.feature.StringIndexer

import org.apache.spark.sql.functions.{col, expr}

import org.apache.spark.sql.DataFrame

object meal {

def main(args: Array[String]): Unit = {

// 实例化spark

val spark = SparkSession.builder()

// 开启hive连接服务

.enableHiveSupport()

// 设置一些 Spark 配置选项

.config("some.spark.config.options", "some.config")

.master("local[3]") // 设置 Spark 运行模式为本地，使用3个线程

.appName("explore") // 设置应用程序名称

.getOrCreate()

// 设置日志输出等级为Error

spark.sparkContext.setLogLevel("Error")

// 获取数据表 "law.law\_visit\_log\_all"

val MealRatingsdata = spark

.read.json("D:\\资料\\1-3周毕业实习\\餐饮大数据智能推荐\\数据\\MealRatings\_201705\_201706.json")

print("菜品评论数据集的行数:")

println(MealRatingsdata.count()) // 输出数据集的行数

print("菜品评论数据集的结构:")

MealRatingsdata.printSchema() // 打印数据集的结构

println("显示原始数据集:")

MealRatingsdata.show(10)

// 剔除任何一个属性有缺失值的行

val MealRatingslostData = MealRatingsdata.na.drop(Seq("UserID", "Rating", "ReviewTime", "Review", "MealID"))

// 计算处理后的数据行数

val lostrowCount = MealRatingslostData.count()

//缺失属性的行数

val lostnum=MealRatingsdata.count()-lostrowCount

println(s"处理的缺失行数: $lostnum")

println(s"处理后的数据行数: $lostrowCount")

// 打印处理后的数据集

println(s"处理后的数据集:")

MealRatingslostData.show(10)

import spark.implicits.\_

// 创建窗口按照 ReviewTime 降序排列

val windowSpec = Window.partitionBy($"UserID", $"MealID").orderBy($"ReviewTime".desc)

// 选择具有最大 ReviewTime 的行

val FinaMealRatingsData = MealRatingslostData.withColumn("row\_number", row\_number().over(windowSpec))

.filter($"row\_number" === 1)

.drop("row\_number")

// 去除重复的行

val processedrepeData = FinaMealRatingsData.dropDuplicates(Seq("UserID", "MealID"))

val ReperowCount = processedrepeData.count()

//缺失属性的行数

val repenum=MealRatingsdata.count()-ReperowCount

println(s"处理的重复行数: $repenum")

println(s"处理的重复数据集：")

processedrepeData.show()

println(s"处理后的数据行数: $ReperowCount")

// 打印处理后的数据集

println(s"处理后的数据集: ")

FinaMealRatingsData.show()//显示最终的菜品评论数据集

// 创建StringIndexer并转换MealID列

val MealIdIndexer= new StringIndexer().setInputCol("MealID").setOutputCol("MealIDIndex")

val indexedData1 = MealIdIndexer.fit(FinaMealRatingsData).transform(FinaMealRatingsData)

// 创建StringIndexer并转换UserID列

val userIdIndexer = new StringIndexer().setInputCol("UserID").setOutputCol("UserIDIndex")

val indexedData2 = userIdIndexer.fit(indexedData1).transform(indexedData1)

//并保存到hive表中

indexedData2.write.mode("overwrite").format("parquet").saveAsTable("rating\_data\_table")

println("转码后的数据集：")

indexedData2.show()

// 查询并显示写入Hive表后的内容

val ratingtableData= spark.sql("SELECT \* FROM rating\_data\_table")

println("写入Hive表rating\_data\_table后的内容：")

ratingtableData.show()

//读取MySQL数据库中的meal\_list表格内容

val MealListData: DataFrame = spark.read

.format("jdbc") //指定数据源格式为JDBC

.option("url", "jdbc:mysql://master:3306") //指定数据库连接URL

.option("driver", "com.mysql.cj.jdbc.Driver") //指定JDBC驱动程序类

.option("dbtable", "data.meal\_list") //指定要读取的数据表名称

.option("user", "root") //指定数据库用户名

.option("password", "123456") //指定数据库密码

.load() //加载数据

print("菜品名数据集的行数:")

println(MealListData.count()) // 输出数据集的行数

print("菜品名数据集的结构:")

MealListData.printSchema() // 打印数据集的结构

MealListData.show(10)//默认只显示20行

//挑选有异常的数据行

val randData = MealListData.filter(col("mealno").isNull || col("mealID").isNull || col("meal\_name").isNull)

// 删除具有任意属性为NULL值的行

val finaMealListData = MealListData

.filter(col("mealno").isNotNull && col("mealID").isNotNull && col("meal\_name").isNotNull)

// 统计处理后的数据行数

val rowCount = finaMealListData.count()

val randnum=MealListData.count()-rowCount

println(s"处理的数据数: $randnum")

println(s"处理的数据集:")

randData.show()

// 显示处理后的数据行数

println(s"处理后的数据数: $rowCount")

println(s"处理后的数据集:")

finaMealListData.show()//显示最终的菜品名单数据集

//进行mealID转码成mealIDIndex

val mealIdIndexer = new StringIndexer().setInputCol("mealID").setOutputCol("mealIDIndex")

val FinaMealListData= mealIdIndexer.fit(finaMealListData).transform(finaMealListData)

//并保存到hive表中

FinaMealListData.write.mode("overwrite").format("parquet").saveAsTable("meal\_list\_table")

println("转码后的数据集：")

FinaMealListData.show()

// 查询并显示写入Hive表后的内容

val mealtableData= spark.sql("SELECT \* FROM meal\_list\_table")

println("写入Hive表meal\_list\_table后的内容：")

mealtableData.show()

// 从 rating\_data\_table 获取 MealIDIndex 列

val mealIDIndexFromRating = ratingtableData.select("MealIDIndex")

// 从 meal\_list\_table 获取 mealIDIndex 和 meal\_name 列

val mealIDIndexAndName = mealtableData.select("mealIDIndex", "meal\_name")

// 将两个数据框连接为 rating\_meal\_data

val ratingMealData: DataFrame = mealIDIndexFromRating

.join(mealIDIndexAndName, Seq("MealIDIndex"), "left")

// 将 rating\_meal\_data 数据框写入 Hive 表（覆盖现有表）

ratingMealData.write.mode("overwrite").insertInto("rating\_meal\_data")

println("写入Hive表后的内容：")

ratingMealData.show()

// 计算每个用户的评论次数，并统计每个用户评论的次数和占比。

val userCount = FinaMealRatingsData.groupBy("UserID").count()

val totalCount = userCount.agg(sum("count")).first().getLong(0)

val userStats = userCount.withColumn("percentage", col("count") / totalCount \* 100)

// 打印每个用户评论的次数和占比

println("每个用户的评论次数和占比：")

userStats.show()

// 从表 "rating\_data\_table" 中读取数据并创建 DataFrame

val ALSData = spark.read.table("rating\_data\_table")

ALSData.show()

// 将数据集划分为训练集和测试集。

val Array(trainingData, testData) = ALSData.randomSplit(Array(0.8, 0.2))

// 将训练集创建并保存到新表MealRatingsDataTrain表中

trainingData.write.mode(SaveMode.Overwrite).saveAsTable("MealRatingsDataTrain")

// 设置ALS模型参数。

val als = new ALS()

.setMaxIter(10)

.setRegParam(0.01)

.setUserCol("UserIDIndex")

.setItemCol("MealIDIndex")

.setRatingCol("Rating")

// 拟合ALS模型。

val model = als.fit(trainingData)

// 使用测试集进行预测，并保存结果到DataFrame。

val predictions = model.transform(testData)

// 评估模型性能，计算均方误差RMSE。

val evaluator = new RegressionEvaluator()

.setMetricName("rmse")

.setLabelCol("Rating")

.setPredictionCol("prediction")

val rmse = evaluator.evaluate(predictions)

println(s"均方根误差（RMSE）: $rmse")

// 11.输出模型的用户因子矩阵和菜品因子矩阵。

val userFactors = model.userFactors

val mealFactors = model.itemFactors

println("用户因子矩阵：")

userFactors.show()

println("菜品因子矩阵：")

mealFactors.show()

// 为所有用户推荐前10个菜品，并输出结果。

val allUsers = ALSData.select("UserIDIndex").distinct()

val allUserRecommendations = model.recommendForUserSubset(allUsers, 10)

println("为所有用户推荐的菜品：")

allUserRecommendations.show(false)

// 展开 recommendations 列

val expandedRecommendations1 = allUserRecommendations.select(

col("UserIDIndex"),

explode(col("recommendations")).as("recommendation")

).select(

col("UserIDIndex"),

col("recommendation.MealIDIndex").alias("MealIDIndex"),

col("recommendation.rating")

)

expandedRecommendations1.show()

// 将 expandedRecommendations 表与 ratingMealData 表连接

val joinedResult1= expandedRecommendations1.join(ratingMealData, Seq("MealIDIndex")).dropDuplicates()

// 打印连接后的结果

println("为所有用户推荐的菜品及对应的菜名：")

joinedResult1.select("UserIDIndex", "MealIDIndex", "meal\_name").show(false)

// 为所有菜品推荐前10个用户，并输出结果。

val allMeals = ALSData.select("MealIDIndex").distinct()

val allMealRecommendations = model.recommendForItemSubset(allMeals, 10)

println("为所有菜品推荐的用户：")

allMealRecommendations.show(false)

// 展开 recommendations 列

val expandedRecommendations2= allUserRecommendations.select(

col("UserIDIndex"),

explode(col("recommendations")).as("recommendation")

).select(

col("UserIDIndex"),

col("recommendation.MealIDIndex").alias("MealIDIndex"),

col("recommendation.rating")

)

// 将 expandedRecommendations 表与 ratingMealData 表连接

val joinedResult2= expandedRecommendations2.join(ratingMealData, Seq("MealIDIndex")).dropDuplicates()

.dropDuplicates()

// 打印连接后的结果

println("为所有菜品推荐的用户及对应的菜名：")

joinedResult2.select("UserIDIndex", "MealIDIndex", "meal\_name").show(false)

// 为部分用户推荐前10个菜品，并输出结果。

val someUsers = ALSData.select("UserIDIndex").distinct().limit(5)

val someUserRecommendations = model.recommendForUserSubset(someUsers, 10)

println("为部分用户推荐的菜品：")

someUserRecommendations.show(false)

val expandedRecommendations3= allUserRecommendations.select(

col("UserIDIndex"),

explode(col("recommendations")).as("recommendation")

).select(

col("UserIDIndex"),

col("recommendation.MealIDIndex").alias("MealIDIndex"),

col("recommendation.rating")

)

// expandedRecommendations 表与 ratingMealData 表连接

val joinedResult3= expandedRecommendations3.join(ratingMealData, Seq("MealIDIndex")).dropDuplicates()

// 打印连接后的结果

println("为部分用户推荐的菜品：")

joinedResult3.select("UserIDIndex", "MealIDIndex", "meal\_name").show(false)

// 为部分菜品推荐前10个用户，并输出结果。

val someMeals = ALSData.select("MealIDIndex").distinct().limit(5)

val someMealRecommendations = model.recommendForItemSubset(someMeals, 10)

println("为部分菜品推荐的用户：")

someMealRecommendations.show(false)

val expandedRecommendations4= allUserRecommendations.select(

col("UserIDIndex"),

explode(col("recommendations")).as("recommendation")

).select(

col("UserIDIndex"),

col("recommendation.MealIDIndex").alias("MealIDIndex"),

col("recommendation.rating")

)

// 将 expandedRecommendations 表与 ratingMealData 表连接

val joinedResult4= expandedRecommendations4.join(ratingMealData, Seq("MealIDIndex")).dropDuplicates()

// 打印连接后的结果

println("为部分菜品推荐的用户：")

joinedResult4.select("UserIDIndex", "MealIDIndex", "meal\_name").show(false)

}

}

## 5.1 创新与优点

①程序实现了针对数据中的缺值、异常、重复这几类异常情况的处理。这是训练得到一个具有较高准确率的模型的必要条件。

②基于现有数据集，程序训练了一个具有一定准确性的模型，使智能推荐策略的实现成为可能。

③为实现智能餐饮大数据推荐，程序提供了四种不同的推荐策略，即为所有用户推荐前10个菜品、所有菜品推荐前10个用户、为部分用户推荐前10个菜品、为部分菜品推荐前10个用户。策略的多样性能够更好地面对复杂多样的应用市场。

使用ALS模型的餐饮大数据智能推荐策略，具有以下优点：

1.提升用户满意度：ALS模型能够根据用户的个性化需求和偏好，为其推荐符合口味和兴趣的菜品。通过提供准确和个性化的推荐，可以增强用户对餐厅的满意度，提高用户体验。

2.增加销售额和利润：通过精准的推荐，餐饮业主可以促使用户购买更多的菜品，提升交叉销售率。这将带来额外的销售收入，增加餐厅的利润。

3.优化库存管理：ALS模型可以分析用户对不同菜品的喜好程度，预测和识别热门菜品和需求较低的菜品。基于这些信息，餐厅可以优化库存管理，减少浪费，提高资源利用效率。

4.降低营销成本：通过智能推荐策略，餐厅可以精确地将特定的促销活动和优惠券等营销策略针对特定用户群体进行推送，提高营销精准度，降低营销成本。

5.数据驱动决策：餐厅通过使用ALS模型分析和挖掘大数据，可以获得更深入的洞察，了解用户喜好、购物习惯以及市场趋势等。这些数据可以为餐厅制定战略决策和经营计划提供有力支持。

6.个性化推荐：ALS模型能够为每个用户定制个性化的推荐结果。它能够捕捉到用户的偏好、口味和习惯，并基于这些信息为用户推荐符合其特定需求的菜品，提供更加个性化的体验

综上所述，使用ALS模型的餐饮大数据智能推荐策略不仅能够提升用户满意度和增加销售额，还能优化库存管理、降低营销成本，并为餐厅提供数据驱动的决策依据，助力餐厅实现业务增长和长期竞争优势

## 5.2 缺点与不足之处

在传统的推荐系统中，通常会采用单一的推荐算法来完成推荐任务。然而，现阶段的推荐系统已经发展为综合性的应用，不再满足于使用单一算法进行推荐。为了更好地为用户提供服务，我们需要融合多个优秀的推荐算法，充分发挥它们各自的优势。

在本设计中，我们选择了ALS算法作为推荐系统的基础。然而，在进一步的研究和学习中，我们将考虑融合其他优秀的推荐算法。通过根据实际需求充分利用每个算法的优势，我们将构建一个更加出色的推荐系统。

**总结**

在完成上述项目实验后，我们对于数据分析和智能推荐领域有了更深入的了解和实践。

项目背景与目标清晰明确：该项目的背景是基于用户评分数据建立菜品推荐模型，以提高老用户的下单率。这个目标非常具体和可操作，使得我们可以有针对性地进行数据处理和模型构建。

数据读取/存储任务：我们需要将用户评分数据集和菜品数据集从不同的源中读取出来，并进行合适的存储。这一过程需要熟悉数据读取和存储的技术框架，例如Hive或Spark SQL，以及对应的存储格式，如JSON或MySQL。

数据探索与预处理任务：在进行智能推荐之前，我们需要对数据进行探索和预处理。这包括了解数据各字段的信息、处理缺失值、异常值和重复值等。对数据进行清洗和处理可以提高模型训练的效果和推荐结果的准确性。

智能推荐任务：根据用户的历史评分数据，我们可以利用机器学习算法或协同过滤等推荐方法来生成个性化的菜品推荐结果。在这个任务中，我们需要熟悉和掌握相关的推荐算法和工具，如Spark MLlib。

实现技术及工具选择：在完成这个项目实验时，我们需要结合目标和数据来选择合适的技术和工具。在本次实训中，我使用了Hive和Spark SQL进行数据读取和处理，同时使用了Spark MLlib进行智能推荐模型的构建。

总的来说，这次实验让我们对于数据分析和智能推荐有了更深入的认识和实践经验。通过实际操作，我们熟悉了数据处理和模型构建的流程，并学会了如何选择合适的技术和工具来完成任务。这对我们的职业发展和提升自己的能力非常有帮助。同时，这次实验也让我深刻理解了数据对于业务决策的重要性，以及智能推荐在提高用户体验和满意度方面的作用。