某团外卖大数据智能推荐系统

目　录
CONTENTS

项目背景描述

- 问题描述
- 整体思路流程

数据预处理

- 原始数据分析
- 异常数据处理
- 数据变换
- 数据分割

建立推荐模型

- 基于用户
- 基于商品
- 基于Spark ASL
- 模型评测

推荐商品

- 向某用户推荐
- 向所有用户推荐

问题描述

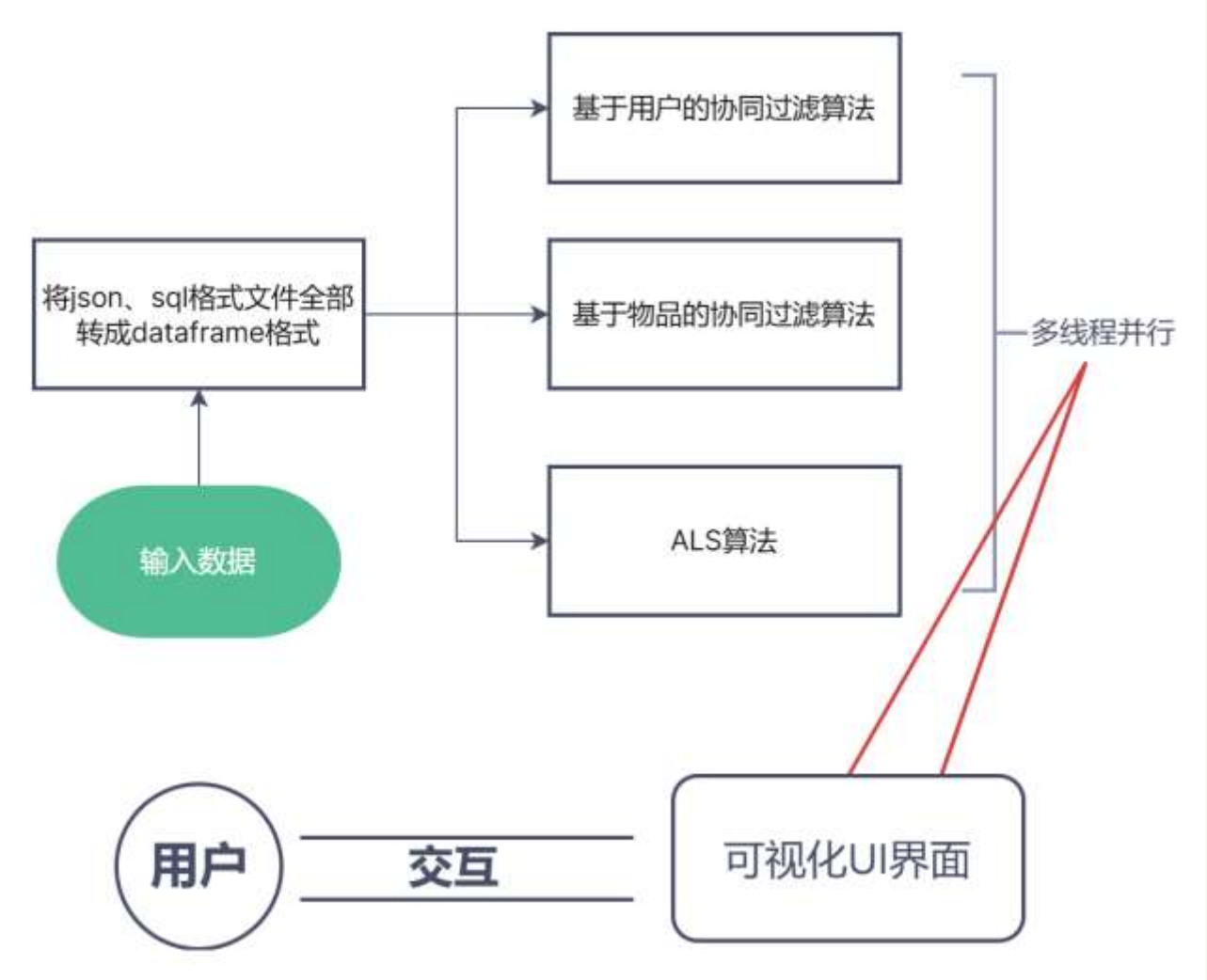
外卖平台面临的问题：W平台订餐完成后，平台会引导用户对于品尝过的菜品进行评价打分。运营方发现老用户的下单率呈现下降态势，希望针对老用户进行个性化的菜品推荐，包括用户的偏爱菜品及新菜品。

本次实验我们主要利用Python技术建立推荐系统模型，通过分析用户的历史行为，主动为用户推荐能够满足他们兴趣和需求的信息，并将长尾产品准确地推荐给所需要的用户。

整体思路流程

本次项目，我们小组选择单机执行python代码进行数据的处理和模型的建立及预测。

在接收到数据后，首先将各类型数据转化为dataframe格式，多线程并行实现三个推荐算法。并创新地制作了**可供用户交互操作的可视化UI界面**，允许用户自由选择不同算法对不同选定用户进行推荐。



数据预处理

数据预处理

读取数据并创建DataFrame：
原始数据是JSON格式存储的，数据结构是固定的，每条记录是由5个属性构成，分别是用户ID、菜品ID、用户评分、用户评论、评论时间戳。因此它非常适合以Python方式来加载，生成DataFrame后进行数据查询。

```
# 读取 SQL 文件
with open('./data/meal_list.sql', 'r', encoding='utf-8') as f:
    sql_content = f.read()
# 提取插入语句中的数据
insert_statements = re.findall(r"INSERT INTO `meal_list` VALUES \((.*?)\);", sql_content, re.S)
# 解析数据
meal_data = []
for statement in insert_statements:
    rows = statement.split("),(")
    for row in rows:
        row_data = row.split(',')
        meal_data.append([int(row_data[0]), row_data[1].strip("'"), row_data[2].strip("'")])
# 创建 DataFrame
meal_df = pd.DataFrame(meal_data, columns=['mealno', 'mealID', 'meal_name'])
# 读取 JSON 文件
with open('./data/MealRatings_201705_201706.json', 'r', encoding='utf-8') as f:
    ratings_data = json.load(f)
# 创建 DataFrame
ratings_df = pd.DataFrame(ratings_data)
```

数据预处理

1.原始数据分析

用户评分数据的探索：数据的分布及其他属性进行统计、针对评分项进行分组统计

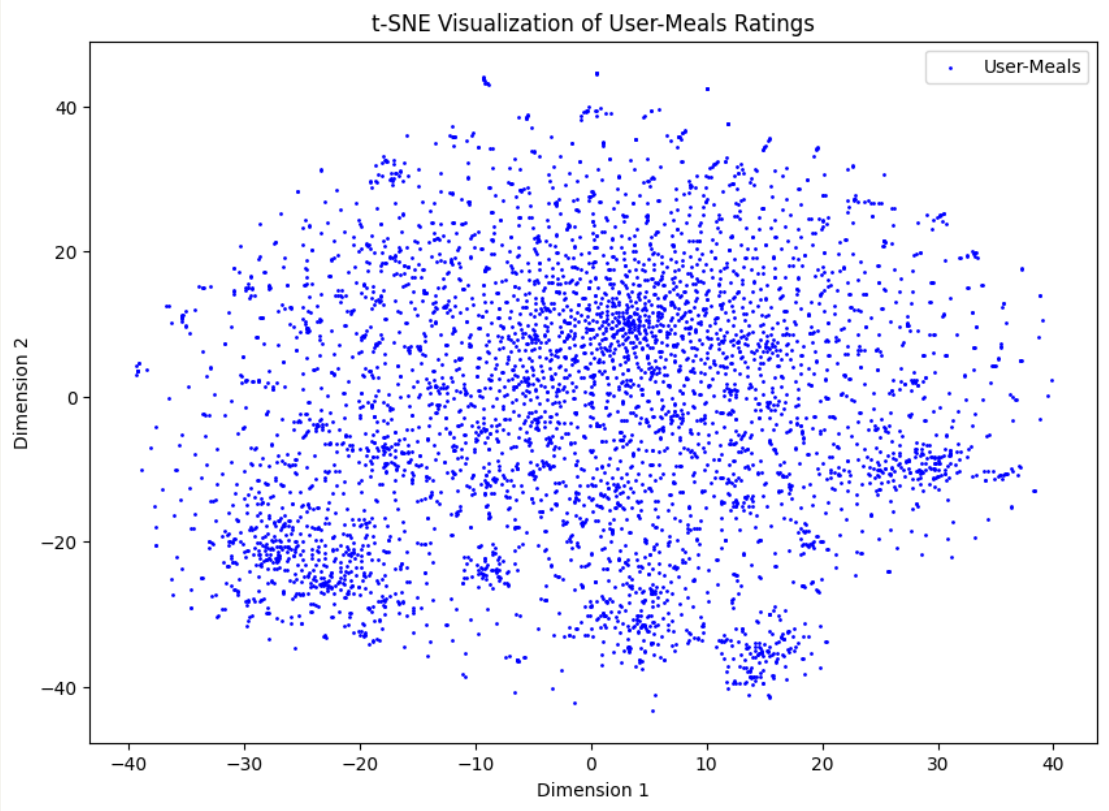
```
总纪录数: 38384, 总用户数: 5130, 总菜品数: 1685, 最高评分: 5.0, 最低评分: 1.0
评分分布:
Rating
5.0    0.547754
4.0    0.238172
3.0    0.116585
2.0    0.051063
1.0    0.046426
```

```python
total_records = len(ratings_df)
total_users = ratings_df['UserID'].nunique()
total_meals = ratings_df['MealID'].nunique()
max_rating = ratings_df['Rating'].max()
min_rating = ratings_df['Rating'].min()

print(f"总纪录数: {total_records}, 总用户数: {total_users}, 总菜品数: {total_meals}, 最高评分: {max_rating},
 最低评分: {min_rating}")

# 评分分布统计
rating_distribution = ratings_df['Rating'].value_counts(normalize=True)
print("评分分布:")
print(rating_distribution)
```

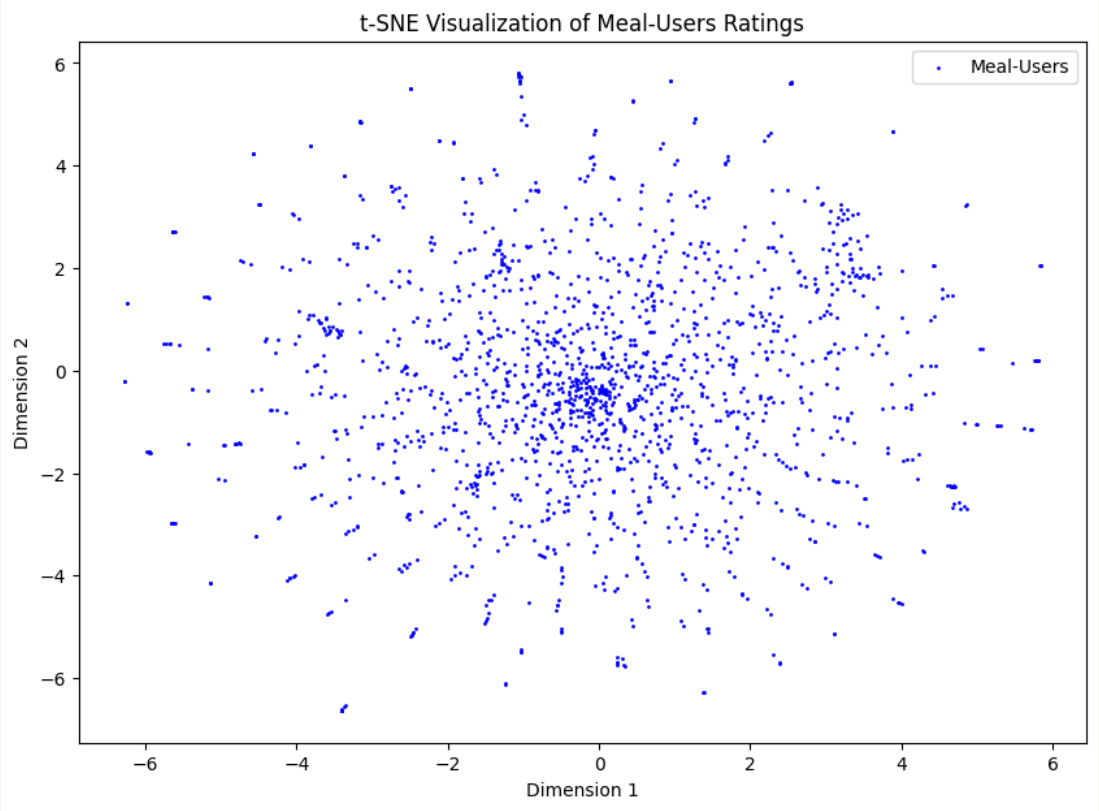
数据预处理

1.原始数据分析

用户评分数据的探索：数据的分布及其他属性进行统计、针对评分项进行分组统计



用户的特征降维分布（呈团块状、相关性弱受大众影响大）



菜品的特征降维分布（呈簇粒状、更精准易区分但多样性不足）

数据预处理

2. 异常数据处理

- 统计是否存在重复评分记录（同一用户对同一菜品多次评分），且输出重复记录总数
- 对于同一用户与菜品的评分，保留最新的评分记录，其他的评分记录不计入。
- 对原始数据集中重复记录进行删除处理，只抽取出各用户对菜品的最新评分记录。

```
# 检查重复评分记录
duplicated_ratings = ratings_df[ratings_df.duplicated(subset=['UserID', 'MealID'], keep=False)]
num_duplicated = len(duplicated_ratings)
print(f"重复评分记录总数: {num_duplicated}")

# 处理重复评分记录
ratings_df.sort_values(by=['UserID', 'MealID', 'ReviewTime'], ascending=[True, True, False], inplace=True)
ratings_df.drop_duplicates(subset=['UserID', 'MealID'], keep='first', inplace=True)

# 更新后的总记录数
total_records_after_dedup = len(ratings_df)
print(f"去重后的记录总数: {total_records_after_dedup}")
```

```
重复评分记录总数: 2518
去重后的记录总数: 37125
```

数据预处理

3. 数据变换

原始的ratings_df DataFrame被扩展了两个新列：user_code和meal_code，分别包含了用户ID和菜品ID的整数编码。

```
# 数据变换
# 编码用户和菜品
user_id_map = {user_id: idx for idx, user_id in enumerate(ratings_df['UserID'].unique())}
meal_id_map = {meal_id: idx for idx, meal_id in enumerate(ratings_df['MealID'].unique())}

ratings_df['user_code'] = ratings_df['UserID'].map(user_id_map)
ratings_df['meal_code'] = ratings_df['MealID'].map(meal_id_map)
```

数据预处理

4. 数据分割

把原始数据按规则分为3部分，分别是：训练集、验证集和测试集。训练集、验证集和测试集的对应占比为80%、10%、10% 。

```
# 数据集分割
train_ratio = 0.8
validation_ratio = 0.1
test_ratio = 0.1

train_df, validation_df, test_df = np.split(ratings_df.sample(frac=1, random_state=42),
                                            [int(train_ratio*len(ratings_df)),
                                             int((train_ratio+validation_ratio)*len(ratings_df))])

print(f"训练集大小: {len(train_df)}, 验证集大小: {len(validation_df)}, 测试集大小: {len(test_df)}")
```

```
训练集大小: 29700, 验证集大小: 3712, 测试集大小: 3713
```

建立推荐模型

1. 基于用户的协同过滤算法

1.通过用户评分数据构建用户-物品矩阵

```
# 创建用户-物品矩阵
user_item_matrix = ratings_df.pivot_table(index='user_code', columns='meal_code', values='Rating')
# 使用pivot_table方法将数据表转换为以用户为行索引，物品为列索引的矩阵形式，矩阵中的值为评分
```

2. 计算用户之间的余弦相似度来找到与目标用户最相似的其他用户

```
# 计算用户之间的相似度
user_similarity = cosine_similarity(user_item_matrix.fillna(0))
# 计算用户-物品矩阵的余弦相似度，对于缺失值用0填充，因为余弦相似度计算中0表示无评分，不影响结果
```

3. 基于这些相似用户的评分数据，为目标用户推荐他们喜欢但目标用户尚未评分的物品

```
# 获取某个用户的评分
def get_user_ratings(user_id):
    return user_item_matrix.loc[user_id].dropna()
# 定义一个函数，根据用户ID返回该用户的评分数据，去除NaN值
```

1. 基于用户的协同过滤算法

4.通过排序和截取前k个推荐结果来呈现最终的推荐列表

```
# 示例推荐
print(user_based_recommendation(0))
# 调用函数，为用户ID为0的用户推荐前4个最可能喜欢的物品
```

```
190    20.784436
89      7.491533
442     7.444267
212     6.120793
```

```python
# 基于用户的协同过滤推荐
def user_based_recommendation(user_id, top_k=4, n_neigh=74):
    # 检查用户是否在用户-物品矩阵中
    if user_id not in user_item_matrix.index:
        return pd.Series()

    # 获取当前用户与其他用户的相似度分数
    user_sim_scores = pd.Series(user_similarity[user_item_matrix.index.get_loc(user_id)], index=user_item_matrix.index)
    # 删除当前用户自身的相似度分数并按相似度降序排序
    user_sim_scores = user_sim_scores.drop(user_id, errors='ignore').sort_values(ascending=False)
    # 选取前n_neigh个最相似的用户
    similar_users = user_sim_scores.index[:n_neigh]

    # 初始化推荐列表
    recommendations = pd.Series(dtype=np.float64)
    for idx, similar_user in enumerate(similar_users):
        # 获取相似用户的评分
        similar_user_ratings = get_user_ratings(similar_user)
        # 根据相似用户的评分和相似度分数计算推荐得分
        recommendations = pd.concat([recommendations, similar_user_ratings * user_sim_scores.iloc[idx]])

    # 按照推荐得分对物品进行分组和求和
    recommendations = recommendations.groupby(recommendations.index).sum()
    # 获取当前用户已评分的物品
    user_rated_items = get_user_ratings(user_id).index
    # 从推荐列表中删除当前用户已评分的物品
    recommendations = recommendations.drop(user_rated_items, errors='ignore')

    # 返回得分最高的top_k个推荐物品
    return recommendations.sort_values(ascending=False).head(top_k)
```

2. 基于物品的协同过滤算法

1.通过转置用户-物品矩阵得到物品-用户矩阵

```
# 创建物品-用户矩阵
item_user_matrix = user_item_matrix.T
```

2.使用cosine_similarity函数计算物品-用户矩阵中物品之间的余弦相似度

```
# 计算物品之间的相似度
item_similarity = cosine_similarity(item_user_matrix.fillna(0))
```

3.对于给定用户，遍历其已评分的每个物品，并基于这些物品的相似物品（忽略用户已评分的相似物品）来累加推荐得分

```
# 获取某个物品的评分
# 定义一个函数，根据物品ID返回所有用户对该物品的评分（忽略NaN值）
def get_item_ratings(item_id):
    return item_user_matrix.loc[item_id].dropna()
```

2. 基于物品的协同过滤算法

3.对于给定用户，遍历其已评分的每个物品，并基于这些物品的相似物品（忽略用户已评分的相似物品）来加权累加推荐得分

```python
# 基于物品的协同过滤推荐
def item_based_recommendation(user_id, top_k=4, n_neigh=78):
    # 检查用户是否在用户-物品矩阵中
    if user_id not in user_item_matrix.index:
        return pd.Series()

    # 获取用户对物品的评分
    user_ratings = get_user_ratings(user_id)
    # 初始化推荐列表
    recommendations = pd.Series(dtype=np.float64)

    for item, rating in user_ratings.items():
        # 检查物品是否在物品-用户矩阵的列中
        if item not in user_item_matrix.columns:
            continue

        # 获取当前物品与其他物品的相似度分数
        item_sim_scores = pd.Series(item_similarity[user_item_matrix.columns.get_loc(item)], index=user_item_matrix.columns).sort_values
        (ascending=False)
        # 选取前n_neigh个最相似的物品
        similar_items = item_sim_scores.index[:n_neigh]

        for idx, similar_item in enumerate(similar_items):
            # 跳过用户已经评分的物品
            if similar_item in user_ratings.index:
                continue
            # 根据相似物品的评分和相似度分数计算推荐得分
            recommendations.at[similar_item] = (recommendations.get(similar_item, 0) + item_sim_scores[similar_item] * rating)

    # 按照推荐得分对物品进行分组和求和
    recommendations = recommendations.groupby(recommendations.index).sum()
    # 返回得分最高的top_k个推荐物品
    return recommendations.sort_values(ascending=False).head(top_k)
```

2. 基于物品的协同过滤算法

4.按得分降序排列推荐结果，并返回前k个推荐物品

```
# 对推荐结果进行分组求和
# 然后按得分降序排列，并返回前k个推荐物品
recommendations = recommendations.groupby(recommendations.index).sum()
return recommendations.sort_values(ascending=False).head(top_k)
```

5.返回用户ID为0的用户的前4个推荐物品

```
# 示例推荐
# 调用item_based_recommendation函数，为用户ID为0的用户推荐前4个最可能喜欢的物品
print(item_based_recommendation(0))
```

```
190     1.208134
1438    0.984844
1538    0.980827
750     0.963872
```

3. 基于ALS的协同过滤算法

1.数据加载与用户-物品矩阵创建：

•读取训练集、验证集和测试集数据。

•创建用户-物品矩阵，行是用户，列是物品，值是评分

```
# 创建用户-物品矩阵
user_item_matrix = train_ratings.pivot_table(
    index='user_code', columns='meal_code', values='Rating')
```

2.相似度计算：计算用户相似度和物品相似度矩阵，使用余弦相似度

```
# 计算用户相似度和物品相似度矩阵
user_similarity = cosine_similarity(user_item_matrix.fillna(0))
item_similarity = cosine_similarity(user_item_matrix.fillna(0).T)
```

3.ALS矩阵分解：定义compute_rmse函数计算预测评分与实际评分之间的RMSE。定义als_train函数使用交替最小二乘法（ALS）进行矩阵分解，得到用户特征矩阵U和物品特征矩阵Vt。

3. 基于ALS的协同过滤算法

```python
# 计算RMSE的函数
def compute_rmse(R, U, Vt):
    R_pred = np.dot(U, Vt)# 计算预测的评分矩阵
    mse = np.sum((R - R_pred) ** 2) / np.count_nonzero(R) # 计算均方误差
    rmse = np.sqrt(mse) # 计算RMSE
    return rmse
```

```python
# ALS训练函数
def als_train(R, k=10, max_iter=10, tol=0.001):
    num_users, num_items = R.shape
    U = np.random.rand(num_users, k)  # 随机初始化用户特征矩阵
    Vt = np.random.rand(k, num_items)  # 随机初始化物品特征矩阵
    R_demeaned = R - np.mean(R, axis=1).reshape(-1, 1)  # 去均值
    for i in range(max_iter):
        # Fix Vt and solve for U
        for u in range(num_users):
            U[u, :] = np.linalg.solve(np.dot(Vt, Vt.T), np.dot(Vt, R_demeaned[u, :].T)).T
        # Fix U and solve for Vt
        for v in range(num_items):
            Vt[:, v] = np.linalg.solve(np.dot(U.T, U), np.dot(U.T, R_demeaned[:, v]))
        rmse = compute_rmse(R_demeaned, U, Vt) # 计算RMSE
        # print(f"Iteration {i+1}/{max_iter}, RMSE: {rmse}")
        if rmse < tol:
            break
    return U, Vt
```

3. 基于ALS的协同过滤算法

4.推荐算法：定义get_user_ratings函数获取某用户的实际评分。定义als_user_recommendation函数根据ALS分解结果为用户推荐前K个评分最高的物品。

```
# 获取某个用户的评分
def get_user_ratings(user_id):
    return user_item_matrix.loc[user_id].dropna()


# ALS推荐算法
def als_user_recommendation(user_id, top_k=4):
    if user_id not in predicted_ratings_df.index:
        return pd.Series()

    user_ratings = predicted_ratings_df.loc[user_id].sort_values(ascending=False)
    return user_ratings.head(top_k)
```

3. 基于ALS的协同过滤算法

5.评估函数：

- 定义calculate_evaluation_score函数计算推荐系统的准确率、召回率和F1分数。
- 计算并打印基于用户相似度、物品相似度和ALS的验证集和测试集的评估结果。

```python
# 计算评估参数
def calculate_evaluation_score(recommendation_func, test_data):
    hits = 0
    recall_total = 0
    precision_total = 0
    for user_id in test_data['user_code'].unique():
        user_test_ratings = test_data[test_data['user_code'] == user_id]
        recommended_items = recommendation_func(user_id).index.tolist()

        for item_id in user_test_ratings['meal_code']:
            if item_id in recommended_items:
                hits += 1
        recall_total += len(user_test_ratings['meal_code'])
        precision_total += len(recommended_items)

    precision_value = hits / precision_total if precision_total > 0 else 0
    recall_value = hits / recall_total if recall_total > 0 else 0

    return (
        precision_value,
        recall_value,
        2 * precision_value * recall_value / (precision_value + recall_value),
    )
```

3. 基于ALS的协同过滤算法

5.评估函数：

- 定义calculate_evaluation_score函数计算推荐系统的准确率、召回率和F1分数。
- 计算并打印基于用户相似度、物品相似度和ALS的验证集和测试集的评估结果。

```
# 计算测试集的准确率
print(
    "User-based CF Validation Evaluation:",
    calculate_evaluation_score(user_based_recommendation, validation_ratings),
)
print(
    "Item-based CF Validation Evaluation:",
    calculate_evaluation_score(item_based_recommendation, validation_ratings),
)
# 计算验证集和测试集的F1分数
print(
    "ALS Validation Evaluation:",
    calculate_evaluation_score(als_user_recommendation, validation_ratings),
)
```

3. 基于ALS的协同过滤算法

6.重新训练模型：将训练集和验证集合并重新训练ALS模型，并再次计算测试集的评估结果

```
# 合并训练集和验证集
combined_train_validation_ratings = pd.concat([train_ratings, validation_ratings])

# 创建新的用户-物品矩阵
user_item_matrix = combined_train_validation_ratings.pivot_table(
    index='user_code', columns='meal_code', values='Rating'
)

# 重新计算用户相似度和物品相似度矩阵
user_similarity = cosine_similarity(user_item_matrix.fillna(0))
item_similarity = cosine_similarity(user_item_matrix.fillna(0).T)
```

后续矩阵分解、计算并打印评估结果的代码同上。

4. 模型评测

对先前的数据分割得到的验证集与测试集上分别运行三个算法，得到的评估指标如下：

考虑到真实情况下用户行为具有较强的随机性，尽管各类指标的数值均在10%附近，我们认为这已经是一个可观的数值。

算法	Precision	Recall	F1
UserBased	0.087109375	0.12015086206896	0.10099637681159
ItemBased	0.083203125	0.11476293103448	0.09646739130434
ALS	0.02828125	0.09752155172413	0.04384689922480

Results of Validation

同时，考虑到上述问题，本小组对测试集结果进行了人工观察，并发现整体推荐结果与用户的过往菜单有较高相似度。

算法	Precision	Recall	F1
ItemBased	0.10500586624951	0.14462698626447	0.12167214229070
UserBased	0.09698865858427	0.13358470239698	0.11238246289792
ALS	0.03269456394211	0.11257743064907	0.05067280882531

Results of Test

推荐商品

1. 向某用户推荐新菜品（可视化代码实现）

代码流程：

1.数据加载

调用 **load_data** 方法读取 SQL 文件并提取插入语句中的数据，创建菜品 DataFrame (meal_df)读取 JSON 文件并创建评分 DataFrame (ratings_df)，处理重复评分记录，按用户和菜品进行排序，并删除重复记录，编码用户和菜品，创建用户编码和菜品编码映射

2.UI 初始化

调用 initUI 方法创建主窗口部件和布局

创建顶部用户选择部分，包括用户下拉框和确定按钮

创建中间部分布局，包括左侧已选菜品部分和右侧推荐部分

创建基于用户的协同过滤、基于物品的协同过滤和ALS推荐算法的推荐分组

3.用户数据加载

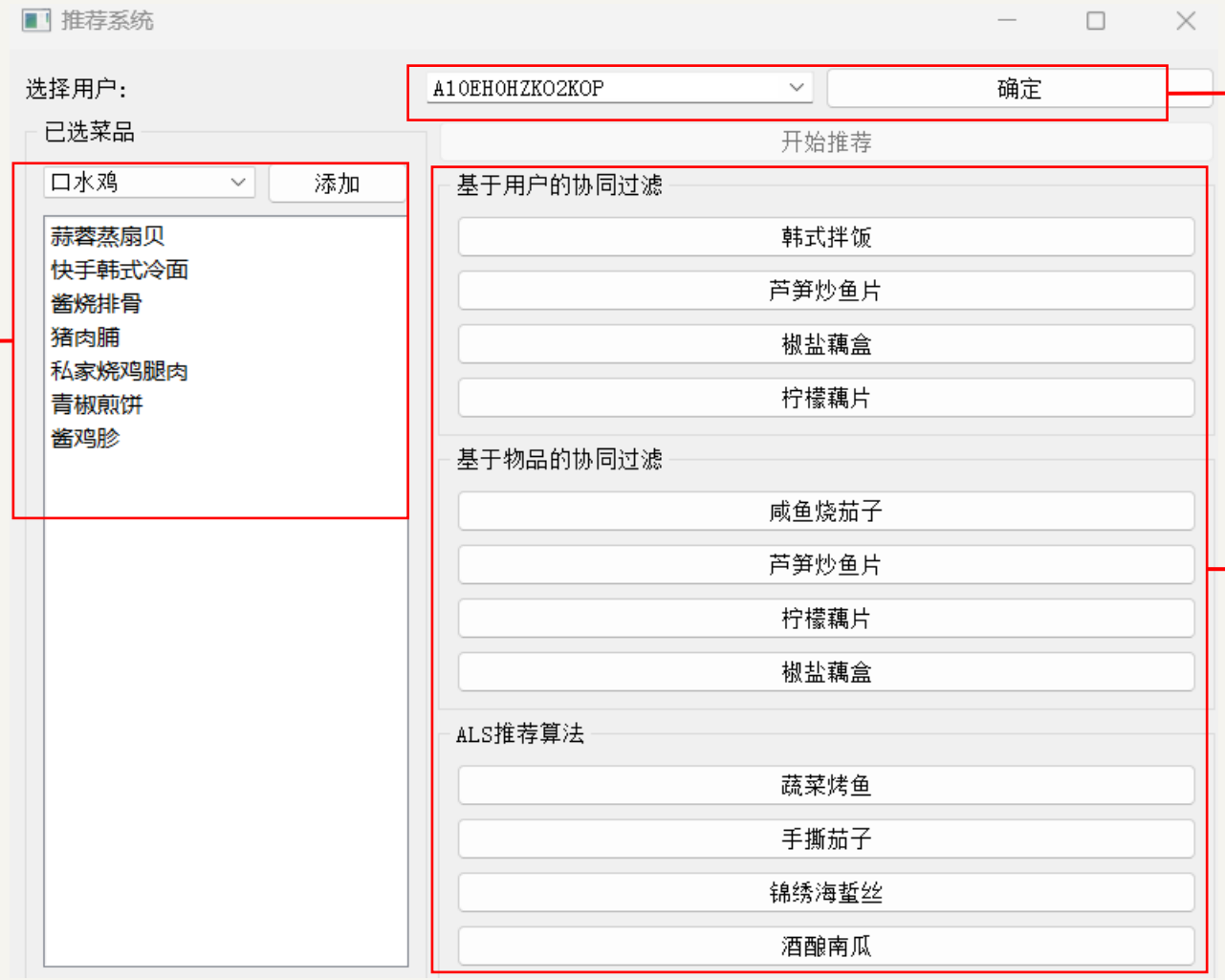
load_user_data 方法，根据选定的用户加载用户数据，并在UI中显示已选菜品

add_meal 方法，向已选菜品列表中添加菜品；remove_meal 方法，从已选菜品列表中删除菜品

4.运行推荐算法

获取已选菜品记录→创建新的用户编码→添加评分记录→创建用户-物品矩阵→计算相似度矩阵→ALS 矩阵分解→基于用户的协同过滤推荐→基于物品的协同过滤推荐→ALS 推荐算法→创建线程来并行运行推荐算法→更新推荐结果到 UI

1. 向某用户推荐新菜品



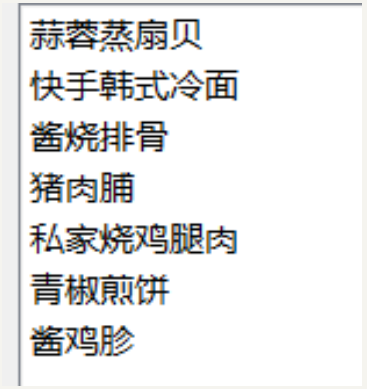
查看该用户已选菜品（训练数据），也可以再添加新菜品数据（视为评分5.0）

指定某一用户

三种推荐算法的推荐结果

1. 向某用户推荐新菜品

基于用户的协同过滤		基于物品的协同过滤		基于ALS的协同过滤	
推荐	相似	推荐	相似	推荐	相似
韩式炒饭	韩式	咸鱼烧茄子	海鲜	蔬菜烤鱼	海鲜
芦笋炒鱼片	海鲜	芦笋炒鱼片	海鲜	手撕茄子	
椒盐藕盒	猪肉	柠檬藕片		锦绣海蛰丝	海鲜
柠檬藕片		椒盐藕盒	猪肉	酒酿南瓜	



1.用户主要喜欢的菜品类型有：海鲜、猪肉、鸡肉、主食（面食）、韩式。
2.基于用户和基于物品的协同过滤与用户喜好相似度较高，有三个相似，较能体现出用户自身喜好，尤其是基于用户的算法推荐的种类也更多样。
3.基于ALS的协同过滤的相似度较低，但给用户推荐了两个素菜，和用户口味较重这一点有所贴合，用户有可能会觉得新鲜与惊喜。

2. 向所有用户推荐新菜品（三种算法比较）

Q1：三种算法的最佳推荐数量的选择
A1：根据三种算法的精确度，召回率和F1值分析有，top_K分别取2（基于用户）、2（基于物品）、5（基于ALS）时算法效果最好。同时考虑的近邻邻居数均近似70取最佳。

Q2：三种算法哪一种算法相对最优？
A2：最佳算法是基于物品的，top_k越大precesion越小、recall越大，综合三种算法来讲top_k=4时总体更加符合大众需求。因此在最后的可视化界面中，我们选择了四个菜品来进行推荐。

Q3：实验亮点
A3：在基础要求上，我们额外实现了菜品推荐的可视化界面，可以快速的对于每个用户的菜品推荐进行响应。还对用户菜品栏添加了，增加和删除菜品来帮助用户实现更加智能且个性化的菜品推荐需求。