数据挖掘互评作业二: 频繁模式与关联规则挖掘 - 网页浏览行为关联规则挖掘

任务: 分析用户在网站上的浏览行为数据，挖掘潜在的跳转规律，为网站提供优化导航结构的建议。

数据集: UCI ML Repository - Anonymous Microsoft Web Data

关联规则挖掘:

1. 数据预处理: 清洗数据，处理缺失值，提取用户浏览记录。
2. 数据探索性分析: 分析最常被访问的页面、页面访问量分布等。
3. 关联规则挖掘: 使用Apriori算法或FP-growth算法，根据用户浏览记录计算频繁项集和关联规则。
4. 结果评估: 计算关联规则的支持度、置信度和提升度，得出强关联规则。
5. 结果分析与应用: 分析得到的关联规则，为网站提供导航结构优化建议，以提升用户体验。

根据提供的信息，这个数据集是通过对www.microsoft.com的日志进行抽样和处理而创建的。该数据记录了38000个匿名随机选择的用户在一周时间内对www.microsoft.com的使用情况。对于每个用户，数据列出了该用户在一周时间内访问的网站的所有区域（Vroots）。

用户仅通过一个顺序号进行标识，例如，用户＃14988，用户＃14989等。该文件不包含任何个人身份信息。共有294个Vroots，它们通过标题（例如“PowerPoint的NetShow”）和URL（例如“/stream”）进行标识。数据来自1998年2月的一周时间段。

每个属性都是www.microsoft.com网站的一个区域（"vroot"）。数据集记录了每个用户在1998年2月的一周时间内访问了哪些Vroots。

接下来，对我所提供的代码进行详细分析:
这段代码是一个用于计算页面相似性矩阵的类。它使用了numpy和scipy库进行向量和距离计算。以下是该类的主要方法和功能:
compute_matrix(): 计算页面相似性矩阵。根据已访问页面的数据，计算每对页面之间的相似性，包括余弦相似度、相关性和杰卡德系数等。计算过程中使用了权重计算和缓存机制来提高效率。
dump_matrix(): 将计算得到的页面相似性矩阵保存到文件中。
hierarchical_cluster(similarities): 对页面相似性矩阵进行层次聚类分析，并计算轮廓系数（silhouette score）来评估聚类质量。
load(file): 从文件中加载页面相似性矩阵。
VectorsPairAnalysis(a_id, a, b_id, b): 页面向量对的分析类，用于计算不同距离度量（余弦相似度、相关性、杰卡德系数）的相似性。
evaluate_allbut1_item_based_recommendation(): 基于项目的推荐算法评估函数。
evaluate_allbut1_user_based_recommendation(): 基于用户的推荐算法评估函数。

接下来，对代码细节进行展开分析

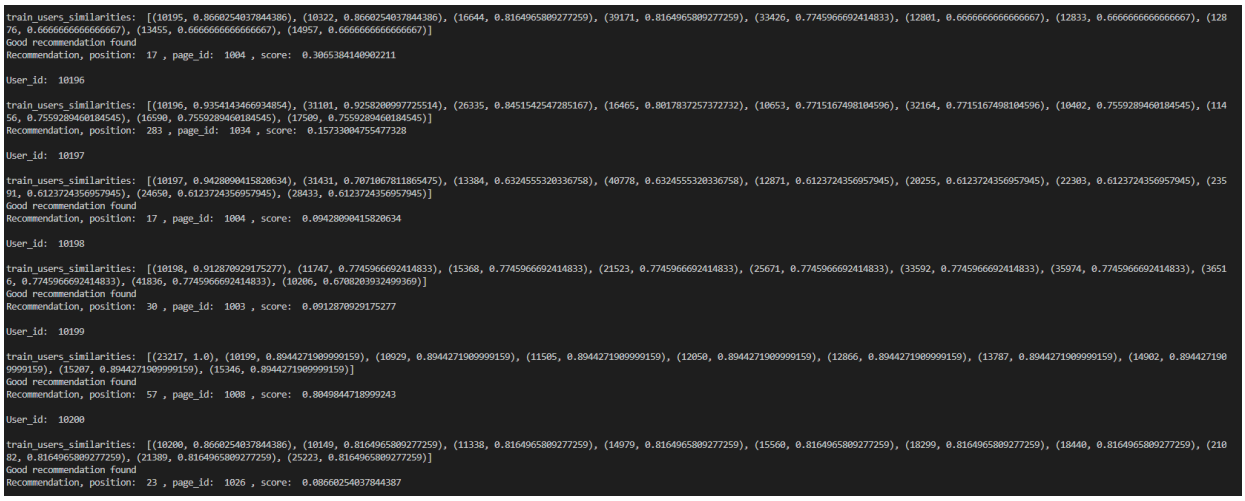
evaluate_allbut1_item_based_recommendation(): 基于项目的推荐算法评估函数。涉及以下数据挖掘相关的步骤:

1. 数据准备:
   函数接收训练数据和测试数据作为输入，包括训练页面数据、训练用户访问页面的ID数据、训练页面的访问数据，以及测试页面数据、测试用户访问页面的ID数据、测试页面的访问数据。
2. 计算页面相似度矩阵:
   创建PagesSimilarityMatrix的实例pcs，传入训练页面数据、训练页面的访问数据和训练用户访问页面的ID数据作为参数。
   调用pcs.compute_matrix()方法计算页面相似度矩阵。

调用pcs.dump_matrix()方法将页面相似度矩阵保存到磁盘。

3. 推荐评估：
   初始化计数器nbr_exact_recommendations用于记录准确推荐的次数。
   获取测试页面的ID列表。
   遍历测试用户的访问页面数据：
   根据测试用户的访问页面数据生成用户的投票情况，将访问过的页面设置为1，未访问的页面设置为0。
   获取用户投票中的赞同位置，即投票为1的位置。
   如果赞同位置数量为1，则跳过该用户。
   应用"All but One"策略，将第一个赞同位置的投票排除。
   遍历用户的投票向量，并尝试找到与赞同页面相似的页面，预测最有可能下次访问的页面。
   计算期望的页面得分，基于页面之间的相似度和投票情况。
   找到最高得分和对应的页面索引作为推荐结果。
   如果推荐的页面索引与排除的投票位置相同，则记录为准确推荐。
   输出推荐结果和得分。
   更新准确推荐次数和考虑用户的数量。

4. 输出结果：
   打印准确推荐的次数和考虑用户的数量。
   总体而言，这段代码利用训练数据中的页面相似度矩阵，根据用户的投票情况预测最有可能被访问的页面。通过"All but One"策略排除其中一个赞同的页面，以评估推荐结果的准确性。



evaluate_allbut1_user_based_recommendation(): 基于用户的推荐算法评估函数。代码涉及以下数据挖掘相关的步骤：

1. 数据准备：
   函数接收训练数据和测试数据作为输入，包括训练页面数据、训练用户访问页面的ID数据、训练页面的访问数据，以及测试页面数据、测试用户访问页面的ID数据、测试页面的访问数据。

2. 用户相似度计算：
   定义了一个get_up_votes_positions函数，用于获取投票为1的位置列表。
   确保测试页面的键与训练页面的键一致。
   初始化准确推荐的次数nbr_exact_recommendations和考虑用户的数量nbr_of_considered_users。
   初始化每个用户的Top N相似用户的数量nbr_top_n。

3. 推荐评估：
   遍历测试用户的访问页面数据：
   根据测试用户的访问页面数据生成用户的投票情况，将访问过的页面设置为1，未访问的页面设置为0。
   获取用户投票中的赞同位置，即投票为1的位置。
   如果赞同位置数量为1，则跳过该用户。
   应用"All but One"策略，将第一个赞同位置的投票排除。
   计算测试用户与所有训练用户之间的相似度。
   根据相似度降序排列训练用户，并保留Top N相似用户。
   输出用户ID和Top N相似用户的信息。
4. 页面推荐：
   遍历用户的投票向量，并尝试找到与赞同页面相似的页面，预测最有可能下次访问的页面。
   如果推荐的页面索引与排除的投票位置相同，则记录为准确推荐。
   输出推荐结果和得分。
5. 输出结果：
   打印准确推荐的次数和考虑用户的数量。
   总体而言，这段代码使用用户之间的相似度来进行推荐。根据测试用户的投票情况和训练用户的历史数据，找到与测试用户相似的训练用户，并根据这些相似用户的行为来预测测试用户下一次可能访问的页面。通过"All but One"策略排除一个赞同的页面，以评估推荐结果的准确性。

```
test_user_id:  10021
Upvotes_positions:  [17, 23, 57, 79, 94, 157, 169, 193, 204, 212, 221, 233, 245, 260, 282, 283, 287]
Excluding position:  17 , vote:  0
Recommendation, position:  2 , page_id:  1289 , score:  1.0


test_user_id:  10022
Upvotes_positions:  [17, 57, 212]
Excluding position:  17 , vote:  0
Recommendation, position:  51 , page_id:  1212 , score:  0.36254232297011785


test_user_id:  10027
Upvotes_positions:  [57, 105, 283]
Excluding position:  57 , vote:  0
Recommendation, position:  239 , page_id:  1283 , score:  1.0


test_user_id:  10030
Upvotes_positions:  [203, 212]
Excluding position:  203 , vote:  0
Recommendation, position:  51 , page_id:  1212 , score:  0.36254232297011785


test_user_id:  10031
Upvotes_positions:  [29, 30, 48, 57, 78, 94, 105, 117, 157, 169, 188, 217, 283, 287]
Excluding position:  29 , vote:  0
Recommendation, position:  2 , page_id:  1289 , score:  1.0


test_user_id:  10034
Upvotes_positions:  [17, 157, 193]
Excluding position:  17 , vote:  0
Recommendation, position:  288 , page_id:  1193 , score:  0.3548732387068153
```

实现Apriori算法或FP-growth算法：

1. 数据预处理：
   从test_pages和train_pages中提取用户浏览记录数据，并将其转换为适合关联规则挖掘的格式。可以将每个用户的浏览记录表示为项集，其中每个项代表一个页面。
   需要创建一个函数来将原始数据转换为项集的形式，以便后续的关联规则挖掘算法使用。例如，可以定义一个名为transform_data的函数，接收pages_visited作为输入，并返回项集列表。
2. 实现Apriori算法：
   编写一个名为apriori的函数，接收项集列表和最小支持度阈值作为输入，并返回频繁项集。
   在apriori函数中，实现Apriori算法的核心步骤，包括生成候选项集、计算支持度、筛选频繁项集等。
   可以使用字典或其他数据结构来存储候选项集和频繁项集，并使用计数来统计每个项集的支持度。
3. 实现FP-growth算法：
   编写一个名为fp_growth的函数，接收项集列表和最小支持度阈值作为输入，并返回频繁项集。
   在fp_growth函数中，实现FP-growth算法的核心步骤，包括构建FP树、构建条件模式基、递归挖掘频繁项集等。
   可以使用递归或其他方法来构建FP树和挖掘频繁项集。
4. 调用算法进行关联规则挖掘：
   根据options.item_based和options.user_based的取值选择要运行的算法。
   如果options.item_based为True，则调用Apriori算法或FP-growth算法进行基于物品的关联规则挖掘。
   将测试数据test_pages和训练数据train_pages作为输入传递给算法，并指定适当的最小支持度阈值。
   如果options.user_based为True，则调用Apriori算法或FP-growth算法进行基于用户的关联规则挖掘。
   将测试数据test_users_visited_pages_ids和训练数据train_users_visited_pages_ids作为输入传递给算法，并指定适当的最小支持度阈值。