**学习进度**

本周学习任务：

（1）了解Part II：Session-Based RSs & Adversarial RSs

（2）代码实践学习：ItemCF、UserCF、NeuMF

下周学习任务：

（1）学习Part III：Evaluating Recommender Systems

（2）代码实践学习

1. **基于会话的推荐系统**
2. **基本认知**

基于会话的推荐系统（Session-Based RSs）以历史会话作为背景，根据用户当前会话中的交互记录，挖掘出特定的短期意图用于定制推荐。Session-Based RSs属于顺序推荐问题，**目标**是预测给定会话中的下一个交互。Session-Based RSs存在**基于会话（Session-based）**和**会话感知（Session-aware）**两种类型，前者针对匿名用户完成基于当前会话的短期意图推荐，后者针对已知用户，在基于当前会话的推荐过程中额外考虑了目标用户的长期偏好。

在Session-Based RSs中，更多时候不知道匿名目标用户的历史会话记录或偏好。**输入**主要依赖基于隐式反馈（用户在会话期间的点击、浏览或购买等行为）的时序上下文，而非基于用户明确特征的传统上下文信息。具体而言，需要输入当前会话中用户的时序交互记录，用于挖掘短期意图完成推荐；也需要输入其他用户的历史交互记录，用于模式挖掘与学习。值得注意的是，针对其他用户的历史交互记录，需根据会话ID或基于Cookie的启发式方法，标注每条记录属于哪个会话。另外，**输出**可以是有序的推荐列表。

至此，可以明确的是，适合Session-Based RSs算法训练的**数据集应当具备的特性**有：交互记录必须按时间顺序组织；交互记录需按会话分组；不一定需要将交互记录映射到用户。然而，**挑战**在于：Session-Based RSs主要依赖当前会话中的交互信息，完成目标短期偏好的挖掘与利用，存在一定程度上的用户冷启动问题，这在会话感知推荐方法中有所缓解。此外，虽然Session-Based RSs更多地依赖隐式反馈作为输入，考虑到了各种类型的交互，从而更充分地利用了数据。但是，多种不同类型的隐式反馈行为本身缺乏统一的解释标准，使得推荐系统在挖掘用户短期模式的过程中更具不确定性。

1. **技术方法**
   1. 无模型方法

基于无模型方法的会话推荐系统，无需从历史会话中学习复杂的数学模型，其主要依赖优化良好的数据挖掘过程，从会话中提取模式用于推荐。相较于依赖模型的会话推荐方法，无模型方法生成的推荐更具解释性（即：可使用从历史会话中提取的模式，解释推荐结果）且在训练数据稀疏时更不容易过拟合。

具体而言，可使用**频繁模式挖掘（Frequent Pattern Mining, FPM）**分析历史交互记录，预先学习不同的用户行为模式（从训练集的历史会话中确定成对物品的共现频率，从而挖掘出类似“购买A的顾客也会购买B”的规则）。然后，分析用户在当前会话中已经执行的操作序列，通过匹配事先学习到的不同规则，用于预测用户的下一步行为。值得注意的是，FPM可以作为更复杂会话推荐方法的基础组件。比如，顺序模式挖掘方法在这种基于共现模式的基础上，额外考虑了商品出现的顺序。

同样，也可以使用**最近邻方法**，完成基于会话的推荐任务。最近邻方法，可以通过离线预计算近邻项目或会话集合的方式，提高在线预测速度。然而，此类方法并没有考虑到会话中项目的顺序问题。具体而言，比如，基于项目的最近邻方法，能够根据当前会话中最后出现项目与其它可选项目间的相似度（项目间的相似度，根据它们在一组训练会话中的共现情况计算而来），完成最近邻项目的推荐。比如，基于会话的最近邻方法，能够根据当前会话最相似的若干近邻会话，计算当前会话对于不同项目的得分，从而完成下一个项目的推荐。相较于前者，基于会话的方法考虑了当前会话中出现的全部项目，而非仅最后一项。

* 1. 基于模型

基于模型的会话推荐系统，能够从用户历史交互序列中学习推荐模型，用于预测未来的行为，属于序列学习方法。相较于前述的无模型方法，此类方法能够从用户历史会话记录中，识别出更复杂的依赖关系和交互。然而，此类方法在训练模型的过程中需要投喂更多的数据，需要花费额外的计算代价。此外，此类方法生成的推荐结果，缺乏可解释性。

具体而言，考虑到会话推荐场景中用户行为的时间顺序性，可以使用基于RNN架构的深度学习模型，完成会话推荐任务。比如，第一个用于会话推荐的GRU4Rec模型。同样，也存在基于CNN、GNN及VAE变分自编码器等架构的深度推荐模型，用于会话推荐任务。特别的，使用强化学习方法构建Session-Based RSs，不仅能额外考虑用户在当前会话中隐含的最终目标和推荐的长期影响，还能通过引入随机性探索的方式，具备打破静态规则的能力。

1. **评估方法**

不仅针对Session-Based RSs，在评估各种类型推荐系统的过程中，主要的评估方法有：离线实验（offline experiments）、用户研究（user studies）和实地测试（field tests）。其中，**离线实验**主要考量推荐系统的性能，比如：预测准确度、覆盖率和流行度等。**用户研究**通过选择若干具有代表性的有限用户来测试使用推荐系统，着重考量推荐结果对于用户实际选择的潜在影响，比如：人机交互的感知质量和系统可用性等。**实地测试**则着重于分析推荐系统在实际应用环境中对于用户的影响。目前为止，在大多数的研究工作中，通过离线实验完成推荐系统的评估更为普遍，用户研究和实地测试相对较少。

值得注意的是，在Session-Based RSs的离线评估环节中，不能使用随机采样的方法来**划分数据集**，这将导致交互记录在时间顺序上的紊乱。更常见的做法是，用户级的分割，即：将一部分用户的数据用于训练，将另一部分用户的数据用于测试，从而确保向新用户提供更有效的推荐。同样，也可以会话为基本单位，进行会话级的数据集分割。此外，还存在交互级的分割方式，即：按照时间节点将交互记录分配到训练或测试集中；或将每个会话中最后几个交互记录放入测试集，其余分配给训练集。显而易见的是，没有普遍适用的划分标准，根据具体情况，甚至可以灵活组合前述方法进行数据划分，从而确保模型的训练和评估效果。比如，可基于会话将数据集划分为训练和测试会话的同时，又可基于交互将测试会话进一步分割为用户概要（旧交互）和测试数据（近期交互）。

1. **研究的局限性与未来方向**

对于整个推荐系统领域的研究而言，**局限性**主要体现在：学术研究的方法论问题和数据的潜在偏差问题。对于前者，主要是指：缺乏统一的评估标准；研究人员经常投入更多精力微调自己新设计的模型，而没有充分关注基线模型的调优工作；没有考虑使用来自不同算法族的简单方法作为基线模型；发表工作缺乏可重复性，代码未开源或开源不完整。对于后者，主要是指：采集而来的数据可能会受到未知因素的影响（比如，部分交互记录可能受到广告或临时折扣的影响），从而造成一定程度上的分布偏差。

由此，考虑改进评估方法，或许是一个可见的**未来方向**。比如，更加关注用户感知对于推荐系统可用性的影响，或者考虑多个利益相关方的视角和数据中潜在偏见的替代离线评估方案。此外，考虑更多类型的数据，或许也是一个可见的未来方向。比如，整合长期偏好、纳入项目元数据和上下文情景或考虑时间因素等。

**二、对抗性推荐系统**

机器学习方法需要保证训练和测试数据分布的一致性，如若机器学习模型的平稳性假设遭到恶意破坏，或许会影响到基于机器学习方法的推荐系统的安全性。

**手工操纵攻击**（hand-crafted shilling attacks）作为一种较常见的攻击类型，通过人为向推荐系统输入虚假交互数据的方式，直接影响某些项目的排名或评分。比如，利用协同过滤算法依赖相似度来完成推荐任务的特性，可以误导系统对特定实例进行错误的预测。然而，这种对于单个用户的攻击缺乏可控性，极有可能会影响到其他用户的推荐结果，甚至影响到整个系统的稳定性。

近年来，伴随着生成对抗网络的发展，**机器学习对抗攻击**（machine-learned adversarial attacks）应运而生。这种方法通过生成微小且精确变化的误导性输入数据，来引导推荐系统产生特定的输出。相较于传统的操纵攻击，这种方法更擅长探索推荐模型的内在弱点，侧重于学习加性扰动。

**三、实践ItemCF**

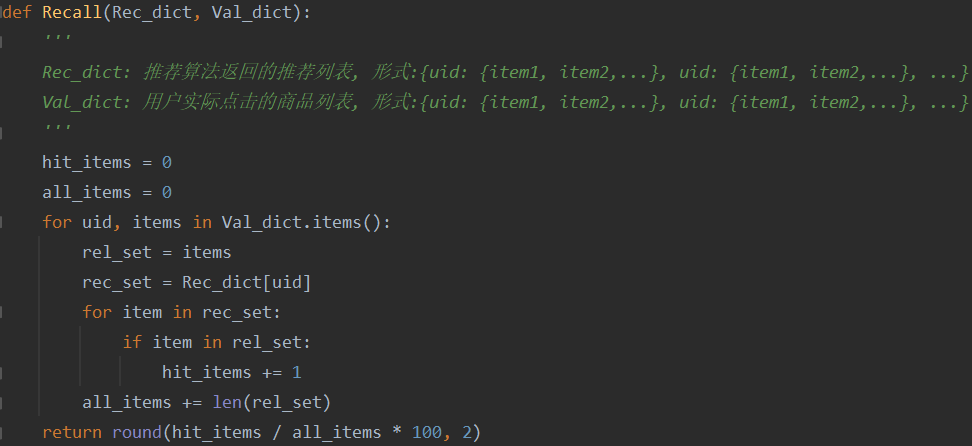
**1. 基本认知**

**基于物品的协同过滤算法，借鉴用户u评价过的且与项目i最近邻的若干项目的评分，用于预测用户u对于目标项目i的评分。**相较于UserCF，此类方法更具解释性，相对缺乏惊喜度。此外，从稳定性的角度出发，需要考虑系统中用户和物品的改变频率和数量，如果用户改变频率更快，则基于物品的方法更稳定。而从准确性和效率的角度出发，则需要考虑系统存储的用户数和物品数比例，当用户远多于物品时，基于物品更准确且计算相似度所需内存和时间往往更少，反之亦然。

1. **评价指标**

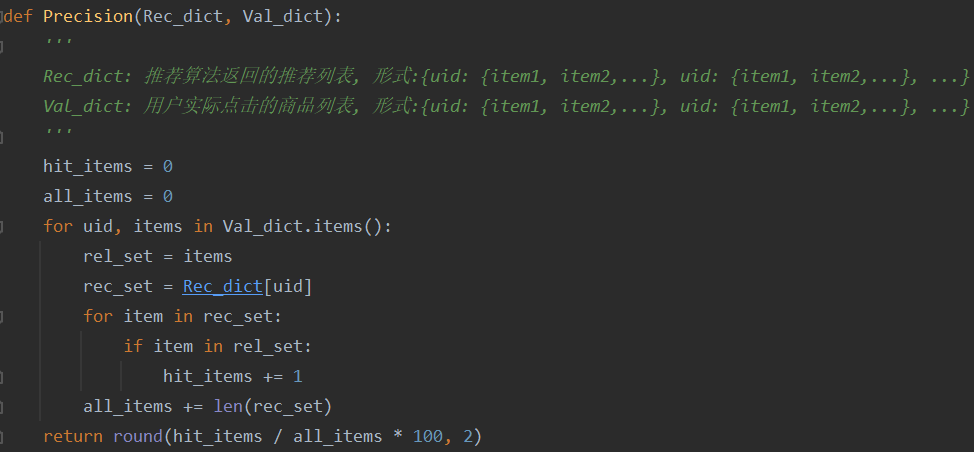
2.1 召回率Recall

**召回率评估总共有多少比例的用户实际点击商品，被命中推荐。**实现方式，如下图所示。具体而言，输入Rec\_dict（推荐算法返回的推荐结果）及Val\_dict（用户实际点击的商品字典），通过遍历每个推荐列表rec\_set中所有推荐商品的方式，累加获得推荐的总命中数hit\_items，从而最终**返回推荐商品总命中数占所有用户实际点击商品的比例。**



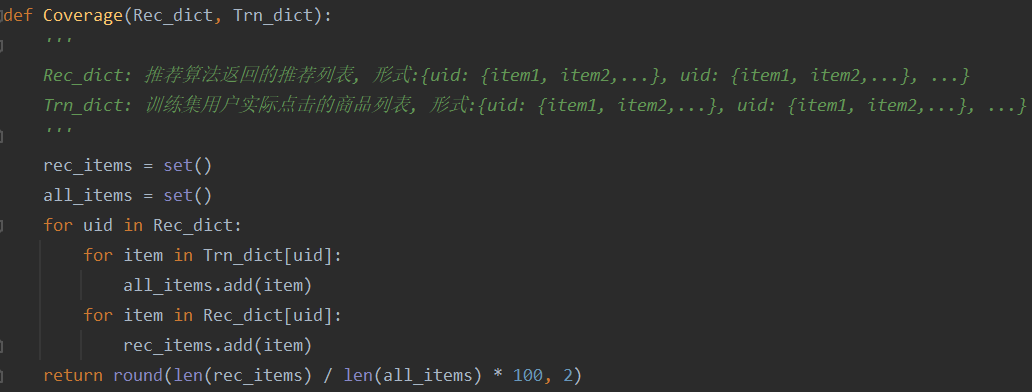
2.2 精确率Precision

**精确率评估总共有多少比例的推荐商品，命中对应用户的实际点击商品列表。**实现方式，如下图所示。具体而言，输入Rec\_dict（推荐算法返回的推荐结果）及Val\_dict（用户实际点击的商品字典），通过遍历每个推荐列表rec\_set中所有推荐商品的方式，累加获得推荐的总命中数hit\_items，从而最终**返回推荐商品总命中数占所有推荐商品数的比例**。



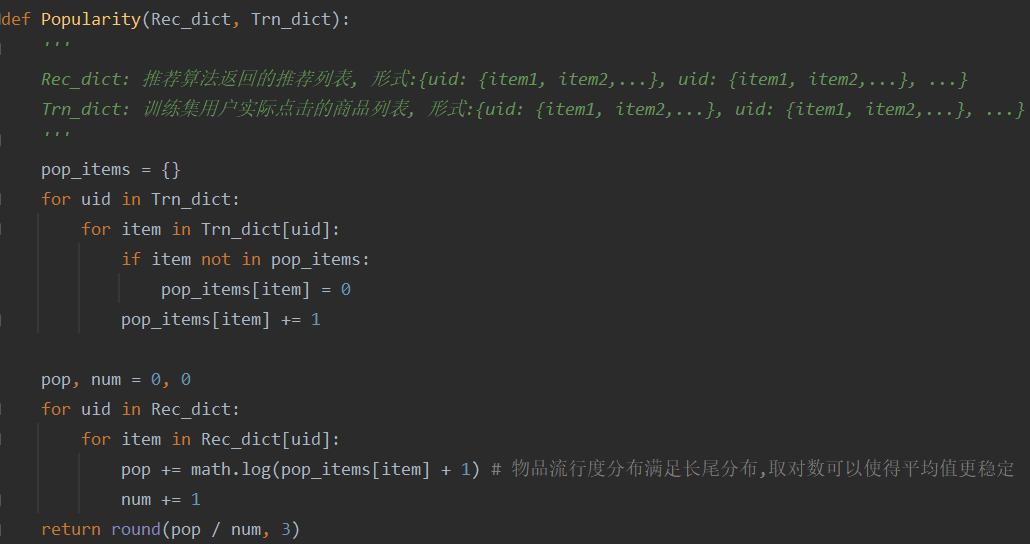
2.3 覆盖率Coverage

**覆盖率评估推荐系统推荐的物品种类，占总物品种类的比例。**实现方式，如下图所示。具体而言，可以依赖集合对象的唯一性，计算所有推荐结果和用户点击字典中的商品种类。显而易见，覆盖率越高则说明越多种类的商品被推荐，内容提供商可能会更关心覆盖率。



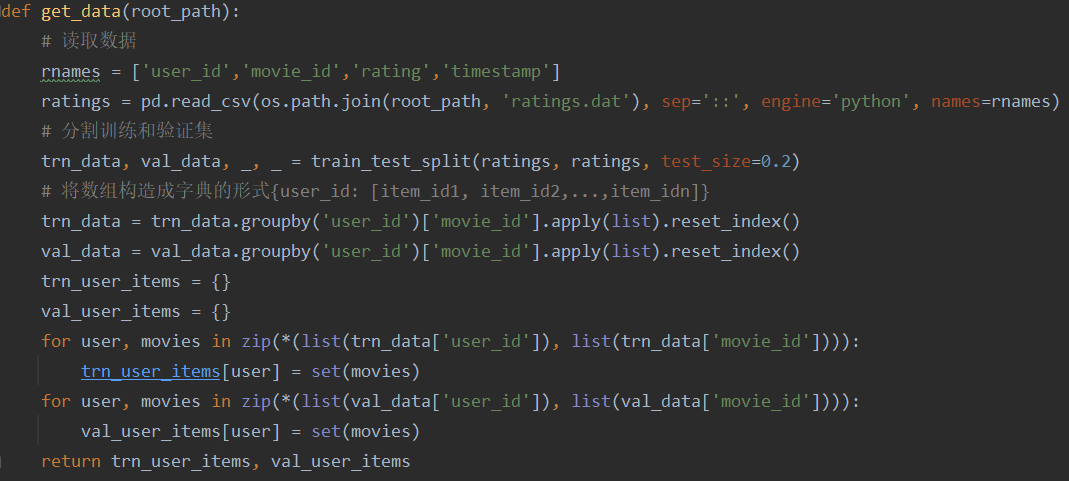
2.4 流行度Popularity

**可以使用平均流行度衡量推荐的新颖性**，平均流行度越高，说明推荐的商品越热门，亦说明推荐的新颖性越低。实现方式如下图所示。具体而言，根据训练集中每种商品的总点击次数，持续累加推荐字典Rec\_dict中每个推荐商品的流行值，并除以累加次数即可算出平均流行度。值得注意的是，每次推荐中某个推荐商品的流行值，可以取Trn\_dict中该商品实际点击次数的对数。



1. **数据处理**

首先使用pd.read\_csv，读取具有用户ID、电影ID、评级及时间戳属性的历史反馈数据；其次使用来自sklearn.model\_selection的train\_test\_split函数，按照4：1的比例分割出训练及验证集；然后使用groupby，根据user\_id分组训练及测试集中的全部记录，并仅记忆分组中每个记录的movie\_id；最后以user\_id为键，以对应分组中包含全部movie\_id的列表为值，构建训练及测试集用户实际点击的商品字典，格式为：{uid1:{item1,item2,...},...}。

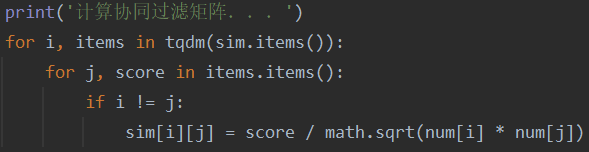
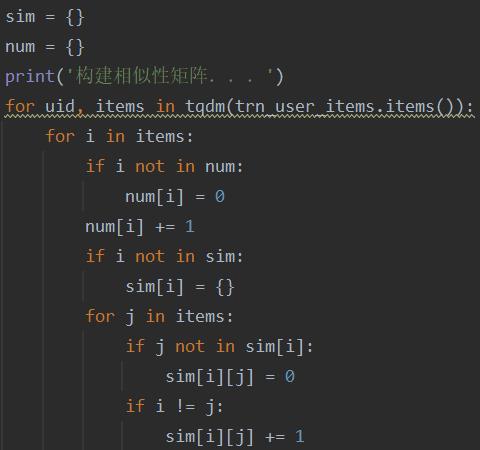


1. **算法实现**

在实现ItemCF算法的过程中，需要接收输入：训练集字典trn\_user\_items、验证集字典val\_user\_items、相似物品数K和推荐商品数N，并根据计算好的物品间相似度矩阵，完成对验证集中每个用户的Top-N推荐。

4.1 构建相似度矩阵

实现方式，如下左图所示。具体而言，需初始化字典sim及num，用于分别表示相似度矩阵及每种商品的总交互次数。随后，循环遍历训练集字典trn\_user\_items中的每个键值对（即：uid1:{item1,item2,...}），并依赖两次内部嵌套循环，统计物品i和物品j的总共现次数及每种物品的遍历次数。值得注意的是，同一种物品间无需统计共现次数，即：无需计算相似度矩阵中主对角线上的相似度。

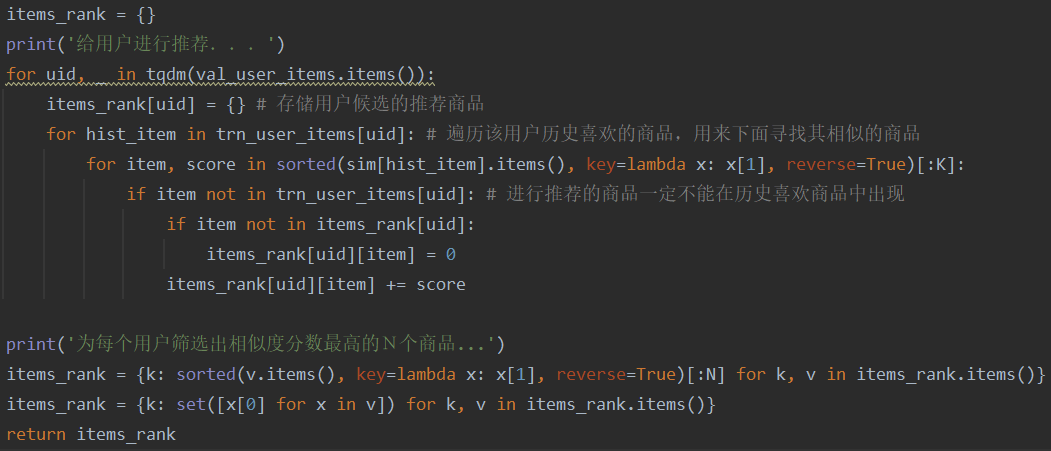


4.2 相似度归一化

显而易见的是，每种商品的总交互次数不同，将导致基于共现次数的物品间相似度计量方法，缺乏统一的比较尺度。由此，可以遍历每对物品i和j间的相似度sim[i][j]，依赖物品i和物品j的总交互次数，完成sim[i][j]的归一化处理。实现方式，如上右图所示。

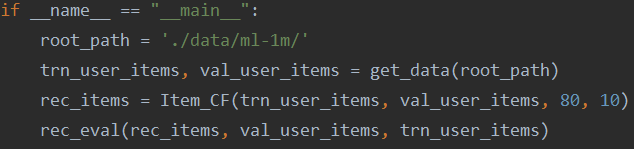
4.3 Top-N推荐

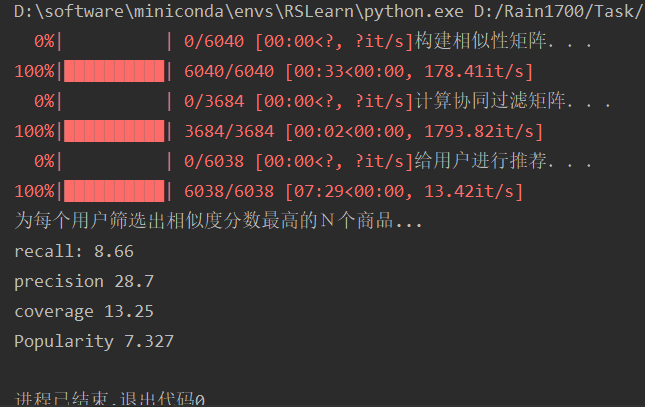
实现方式，如下图所示。首先，循环遍历验证集中的每个键值对，嵌套遍历值对象中的每个物品hist\_item。其次，依赖相似度矩阵sim，获取同物品hist\_item最相似的K个近邻物品及其相似度分数。然后，以用户ID为items\_rank的键，以物品ID为值字典对象中的键，累加每个近邻物品的相似度分数（若物品在用户历史交互中出现过，则不考虑该近邻物品）。最后，根据值字典对象中的累加分数，重排items\_rank中每个用户键对应值字典对象内的近邻物品，从而为每个用户筛选出相似度累加分数最高的N个商品。



4.4 实验结果

实验结果，如下图所示。



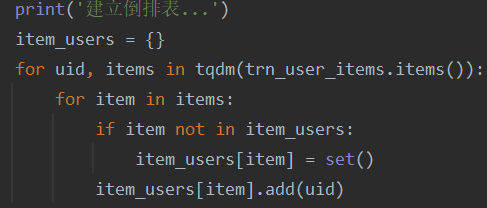


1. **实践UserCF**

**基于用户的协同过滤算法，借鉴用户u最近邻的部分用户对于项目i的评分，用于预测用户u对于项目i的评分。**UserCF的评估方法和数据预处理过程，雷同前述ItemCF。由此，仅对UserCF的算法实现部分进行解释，具体如下。

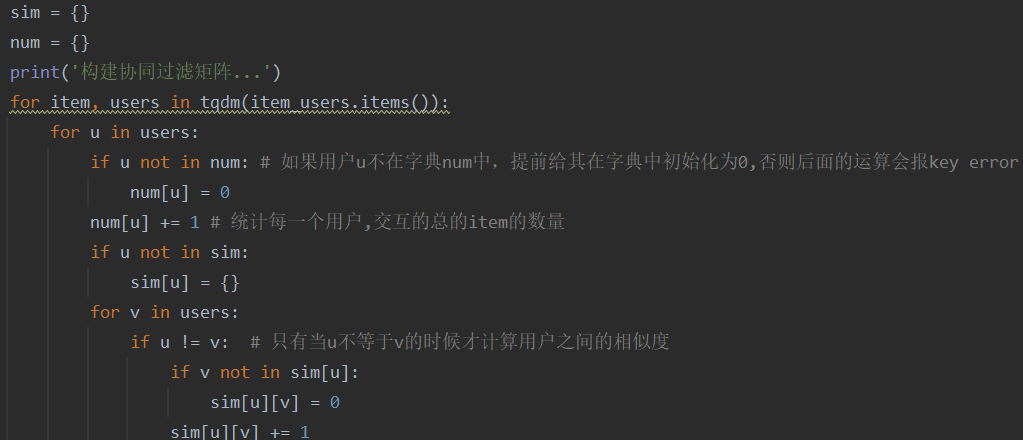
1. **建立倒排表**

将格式为{user\_id1:{item\_id1, item\_id2, ...}, ...}的训练集数据trn\_user\_items，倒排转化为{item\_id1:{user\_id1, user\_id2, ...}, ...}格式的item\_users。



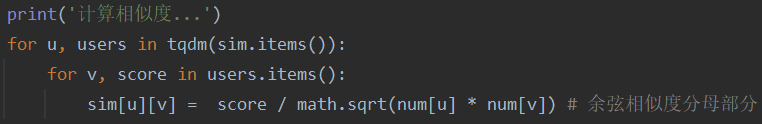
1. **构建相似度矩阵**

实现方式如下图所示。具体而言，需初始化字典sim及num，用于分别表示用户间共同交互过的商品数量和每个用户的商品交互次数。随后，循环遍历倒排训练集item\_users中的每个键值对（即：item\_id1:{user\_id1,user\_id2,...}），并依赖两次内部嵌套循环，统计用户u和用户v的总共现次数和每个用户的商品交互次数。值得注意的是，同一用户间不需要统计共现次数，即：无需计算相似度矩阵中主对角线上的相似度。



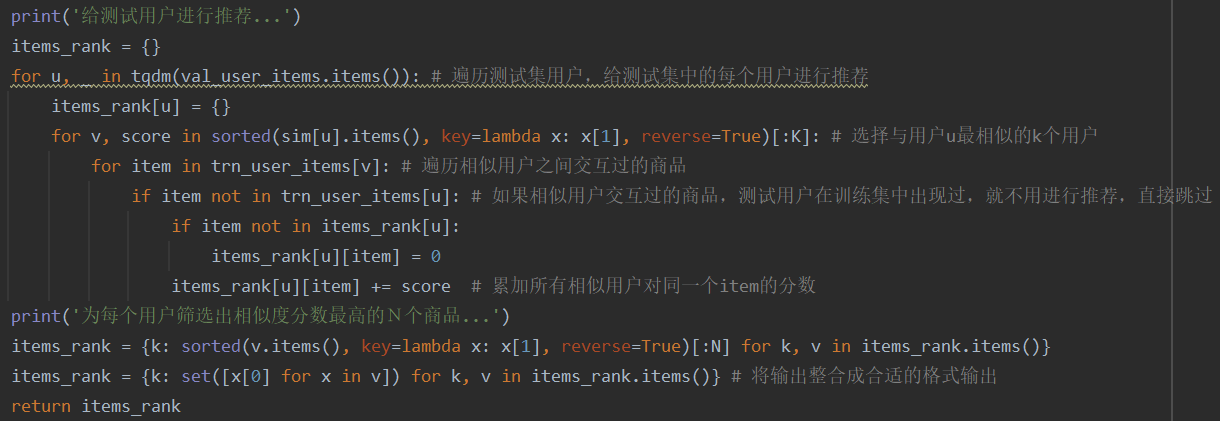
1. **相似度归一化**

根据在不同物品中用户间的总共现次数，完成相似度初步计算，相当于余弦相似度计算公式中的分子部分。至此，还需完成相似度的归一化处理，即实现余弦相似度计算公式中的分母部分。实现方式，如下图所示。具体而言，可以遍历每对用户u和v间的相似度sim[u][v]，依赖用户u和用户v的商品交互次数num[u]及num[v]，完成sim[u][v]的归一化处理。



1. **Top-N推荐**

实现方式，如下图所示。首先，循环遍历验证集val\_user\_items中的每个用户u。其次，依赖相似度矩阵sim，获取同用户u最相似的K个近邻用户v及其相似度分数。然后，以用户u为items\_rank的键，以不同物品为值字典对象中的键，累加所有近邻用户对同一物品的相似度分数（无需考虑在用户u的历史交互中出现过的物品）。最后，根据值字典对象中的累加分数，重排items\_rank中每个用户键对应值字典对象内的物品，从而为每个用户筛选出相似度累加分数最高的N个推荐物品。

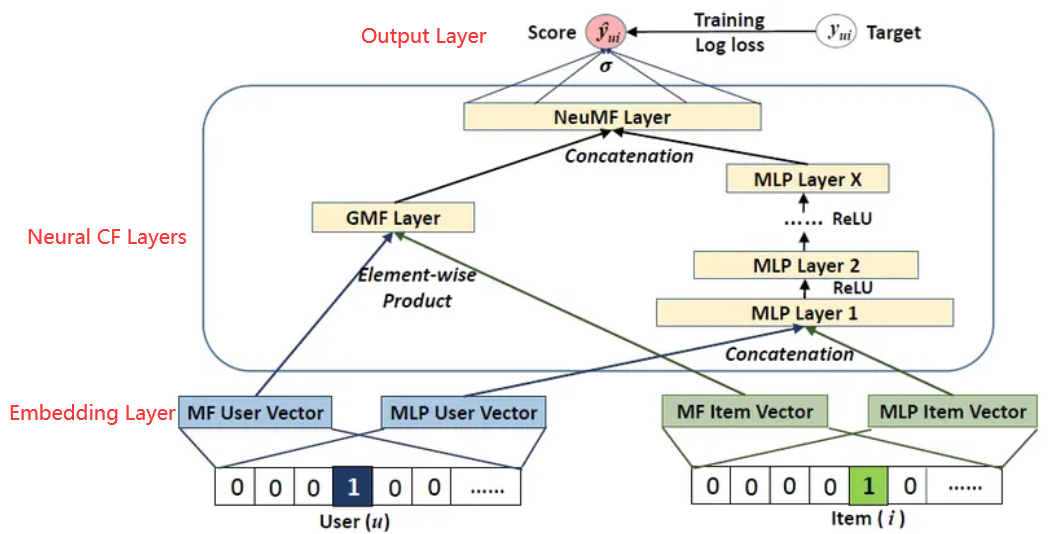


**五、NeuralCF**

**1. 基本认知**

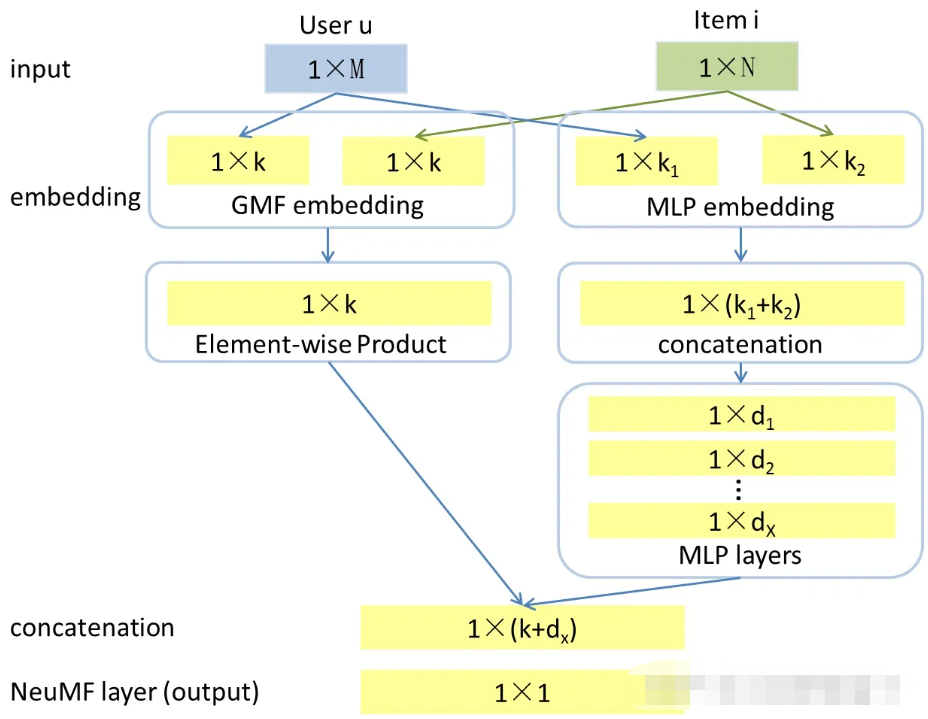
如前所述，传统协同过滤算法，直接利用稀疏的共现矩阵完成推荐预测。此类方法的泛化能力较弱，尤其遇到历史交互非常少的用户时，将存在严重的冷启动问题。随后，学者们利用深度学习改进传统协同过滤方法，提出了神经网络协同过滤算法NeuCF，极大提高了推荐模型的泛化能力。**NeuralCF主要有GMF、MLP及NeuMF三种实现方式，区别在于NeuralCF通用框架中的Neural CF Layers不同。**由于NeuMF模型的Neural CF Layers设计结合了GMF和MLP，因此本节主要解释NeuMF模型的实现方式。

NeuMF模型结构，如下图所示。



其中，**嵌入层**将表示用户或物品的独热编码映射为低维的嵌入特征向量，嵌入表示空间可理解为一种潜在因素空间，嵌入表示的每个维度都能够隐含地表达用户或物品某一方面的特征因素。值得注意的是，原文作者认为：在最优调参的情形中，GMF和MLP的嵌入维数不一定相同，甚至相差很大。由此，在NeuMF模型训练的过程中，需要分别学习GMF和MLP的嵌入层参数。更进一步，GMF通过逐元素相乘的方式，**合并用户和物品的嵌入表示**，要求用户和物品嵌入维数相同。MLP则直接将用户和物品嵌入表示拼接起来，不要求用户和物品的嵌入维数相同。此后，**将分别经过GMF Layer和MLP处理的不同输出向量拼接**起来，投入到最后的NeuMF layer输出层中，即可完成预测。

NeuMF模型的输入输出尺寸，如下图所示。



值得补充的是，不仅可以将表示某个特定用户或物品的独热编码作为原始输入，还可以通过重构某用户或物品原始输入向量的方式，额外考虑上下文，从而捕捉更丰富的特征信息和隐藏关系。比如，在表示某用户的原始输入向量尾部，额外拼接一个考虑用户年龄或偏好等属性的特征向量。

1. **数据预处理**

下载电影评分数据集MovieLens-1-Million（ml-1m），获得具有用户ID、物品ID、评分及时间戳等属性的数据文件train.rating及test.rating，还需要获得数据文件test.negative，用于存储同test.rating中每个样本记录相对应的若干负例物品（该用户没有评分过的物品）子列表。原始数据采用稀疏的方式存储，由此需要将前述三个必需的数据文件预处理为矩阵的形式组织和存储。由于篇幅有限，此处省略预处理环节的实现方式，详情请见：

[hexiangnan/neural\_collaborative\_filtering: Neural Collaborative Filtering (github.com)](https://github.com/hexiangnan/neural_collaborative_filtering)

针对预处理好的矩阵数据文件：train.npy、testRatings.npy及testNegatives.npy。其中，二维矩阵数据**train.npy**用于模型训练，行表示用户索引，列表示物品索引，值域为0或1（若某用户对某物品评过分则取值为1，否则取值为0）。**testRatings.npy**则相当一个二维嵌套列表，每个元素对应一个特定的用户，元素的内容为：[用户索引, 物品索引]。在测试环节，testRatings.npy中每个列表元素的子元素，分别作为一次推荐预测的服务对象和结果标签。值得注意的是，testRatings.npy中某用户u的标签物品i，在train.npy中对应(u,i)的取值应该为0。而在二维嵌套列表**testNegatives.npy**中，其每个元素同样对应一个特定的用户，元素内容为：[该用户没有评过分的物品1，没有评过分的物品2，...]。

值得补充的是，需要使用来自torch.utils.data模块的TensorDataset及DataLoader函数，完成**数据迭代器**的定义，用于分批输出特定数量的样本数据（[用户ID，物品ID]）及标签（若用户对该物品评过分，则标签取值为1，否则为0）。实现方式如下：

train\_dataset = TensorDataset(torch.tensor(train\_x), torch.tensor(labels).float())  
dl\_train = DataLoader(train\_dataset, batch\_size=batch\_size, shuffle=True)

1. **NeuMF实现类**

实现方式及解释，如下所示。其中，GMF和MLP部分的嵌入层，需分开完成独立学习。然而，为了简化实现过程，它们的嵌入表示维数相同。此外，GMF部分只需通过逐元素相乘的方式结合用户和物品的嵌入表示，即可获得GMF部分的最终输出。而后，拼接GMF和MLP部分的输出，并输入到NeuMF模型的输出层，即可完成最终的非线性映射学习。

class NeuralMF(nn.Module):  
 def \_\_init\_\_(self, num\_users, num\_items, mf\_dim, layers):  
 super(NeuralMF, self).\_\_init\_\_()  
 # GMF部分的用户和物品Embedding层  
 self.MF\_Embedding\_User = nn.Embedding(num\_embeddings=num\_users, embedding\_dim=mf\_dim)  
 self.MF\_Embedding\_Item = nn.Embedding(num\_embeddings=num\_items, embedding\_dim=mf\_dim)  
 # MLP部分的用户和物品Embedding层  
 self.MLP\_Embedding\_User = nn.Embedding(num\_embeddings=num\_users, embedding\_dim=layers[0]//2)  
 self.MLP\_Embedding\_Item = nn.Embedding(num\_embeddings=num\_items, embedding\_dim=layers[0]//2)  
 # MLP部分的隐藏及输出层  
 self.dnn\_network = nn.ModuleList([nn.Linear(layer[0], layer[1]) for layer in list(zip(layers[:-1], layers[1:]))])  
 self.linear = nn.Linear(layers[-1], mf\_dim)  
 # NeuMF模型的最终输出层  
 self.linear2 = nn.Linear(2\*mf\_dim, 1)  
 self.sigmoid = nn.Sigmoid()  
 def forward(self, inputs):  
 inputs = inputs.long()  
 MF\_Embedding\_User = self.MF\_Embedding\_User(inputs[:, 0]) # GMF部分的用户和物品的embedding  
 MF\_Embedding\_Item = self.MF\_Embedding\_Item(inputs[:, 1])  
 mf\_vec = torch.mul(MF\_Embedding\_User, MF\_Embedding\_Item) # 将用户和物品的嵌入表示逐元素相乘  
 MLP\_Embedding\_User = self.MLP\_Embedding\_User(inputs[:, 0]) # MLP部分的用户和物品的embedding  
 MLP\_Embedding\_Item = self.MLP\_Embedding\_Item(inputs[:, 1])  
 x = torch.cat([MF\_Embedding\_User, MF\_Embedding\_Item], dim=-1) # 拼接用户和物品的嵌入表示  
 for linear in self.dnn\_network: # 依赖循环完成MLP部分的，多层线性层传递学习  
 x = linear(x); x = F.relu(x)  
 mlp\_vec = self.linear(x)  
 vector = torch.cat([mf\_vec, mlp\_vec], dim=-1) # 拼接GMF和MLP部分的输出  
 linear = self.linear2(vector) # NeuMF模型的最终输出层  
 output = self.sigmoid(linear)  
 return output

1. **评估方法**

实现方式及解释，如下所示。其中，选用**评价指标Hit Ration及NDCG**，针对用户的Top-N推荐结果，用于分别评估标签物品gtItem是否命中及推荐位次。具体而言，若针对某用户的Top-N推荐列表中，存在testRating中对应的标签物品，则该用户的HR取值为1。同时，若标签物品在推荐列表中的位置越靠前，则NDCG取值越大，即：额外考虑了标签物品在某用户Top-N推荐列表中的命中排名。

更进一步，针对testRating中的每个用户，需要根据testNegatives中对应用户索引处的列表元素中所有物品（包含testRating中目标用户的标签物品gtItem），预测出有序的N个最佳推荐物品，从而完成针对该用户的Top-N推荐任务。此后，计算当前批次中每个目标用户的Hit Ration及NDCG值，取平均值即可完成小批次数据的评估工作。

\_model = None; \_testRatings = None; \_testNegatives = None; \_K = None  
def getHitRatio(ranklist, gtItem): # Hit Ration  
 for item in ranklist:  
 if item == gtItem:  
 return 1  
 return 0  
def getNDCG(ranklist, gtItem): # NDCG（归一化折损累计增益）  
 for i in range(len(ranklist)):  
 item = ranklist[i]  
 if item == gtItem:  
 return np.log(2) / np.log(i+2)  
 return 0  
# 对testRatings中的一个用户样本进预测和评估  
def eval\_one\_rating(idx):  
 rating = \_testRatings[idx]; items = \_testNegatives[idx]  
 u = rating[0]; gtItem = rating[1] # 用户及标签物品ID  
 items.append(gtItem) # 将标签物品放入负样本列表中，用于构建测试样本集  
 map\_item\_score = {} # 存储物品和对应的预测分数  
 users = np.full(len(items), u, dtype='int32') # 创建与负样本数量相同的用户ID数组  
 test\_data = torch.tensor(np.vstack([users, np.array(items)]).T).to(device) # 构建测试数据，并转换为PyTorch张量  
 predictions = \_model(test\_data) # 使用模型进行预测  
 for i in range(len(items)): # 将预测结果存储到map\_item\_score字典中  
 item = items[i]; map\_item\_score[item] = predictions[i].data.cpu().numpy()[0]  
 items.pop() # 移除添加的标签物品  
 ranklist = heapq.nlargest(\_K, map\_item\_score, key=lambda k: map\_item\_score[k]) # 基于堆排序获取前K个推荐物品

# 计算Hit Ratio和NDCG评估值  
 hr = getHitRatio(ranklist, gtItem)

ndcg = getNDCG(ranklist, gtItem)  
 return hr, ndcg  
# 整体上评估模型性能  
def evaluate\_model(model, testRatings, testNegatives, K):  
 global \_model; global \_testRatings; global \_testNegatives; global \_K  
 \_model = model; \_testNegatives = testNegatives; \_testRatings = testRatings; \_K = K  
 hits, ndcgs = [], [] # 存储所有评分的Hit Ratio和NDCG  
 for idx in range(len(\_testRatings)): # 遍历所有测试评分  
 (hr, ndcg) = eval\_one\_rating(idx) # 对每个评分进行评估  
 hits.append(hr); ndcgs.append(ndcg)  
 return hits, ndcgs

1. **模型训练与测试**

实现方式及解释，如下所示。考虑到模型训练与测试需要花费的时间，本次仅进行一个迭代周期的实验。其中，在分批次更新模型参数的过程中，需要依赖前述定义好的评价函数，用于评估新参数在测试集上的性能，并持续记忆最佳测试性能下的模型参数，保存为ml-1m\_NeuralMF.pkl文件。

best\_hr, best\_ndcg, best\_iter = hr, ndcg, -1 # 初始化最佳HR、NDCG和迭代次数  
epochs = 1; log\_step\_freq = 10000  
loss\_func = nn.BCELoss(); optimizer = torch.optim.Adam(params=model.parameters(), lr=lr)  
for epoch in range(epochs):  
 # 进入训练模式  
 model.train()  
 loss\_sum = 0.0 # 积累每个batch的损失总和  
 for step, (features, labels) in enumerate(dl\_train, 1): # 遍历训练数据迭代器  
 features, labels = features.cuda(), labels.cuda()  
 optimizer.zero\_grad()  
 predictions = model(features)  
 predictions = predictions.squeeze(1)  
 loss = loss\_func(predictions, labels)  
 loss.backward()  
 optimizer.step()  
 loss\_sum += loss.item() # 累计损失值  
 if step % log\_step\_freq == 0: # 若达到日志打印频率log\_step\_freq，则在控制台打印损失  
 print(("[step = %d] loss: %.3f") % (step, loss\_sum/step))  
 # 进入评估模式  
 model.eval()  
 (hits, ndcgs) = evaluate\_model(model, testRatings, testNegatives, topK) # 使用测试集评估  
 hr, ndcg = np.array(hits).mean(), np.array(ndcgs).mean() # 计算平均HR和NDCG  
 if hr > best\_hr: # 如果当前批次的平均HR超过历史最佳HR  
 best\_hr, best\_ndcg, best\_iter = hr, ndcg, epoch # 更新最佳HR、NDCG和迭代次数  
 torch.save(model.state\_dict(), 'Pre\_train/m1-1m\_NeuralMF.pkl') # 保存模型参数  
 info = (epoch, loss\_sum/step, hr, ndcg) # 打印本次迭代周期的loss、hr、ndcg  
 print(("\nEPOCH = %d, loss = %.3f, hr = %.3f, ndcg = %.3f") %info)

1. **实验结果**

本次关于实现NeuMF模型的实验结果，如下图所示。

