标点符

推荐算法之贝叶斯个性化排序 BPR

2019-07-23 · 2 min read

就像哲学有不同的流派一样，推荐系统的算法设计思路也可以分为不同的流派。排序学习恰恰就是其中的一种流派。熟悉 RecSys 等推荐系统国际会议的从业者可能会发现，自 2010 年以后的若干年内，陆续出现了许多基于排序学习的推荐系统算法。从 Bayesian Personalized Ranking (BPR) 到后续的 Collaborative Less is More Filtering (CLiMF) 以及 GapFM 和 XCLiMF 等算法，在推荐系统领域出现了百家争鸣，百花齐放的局面。今天主要学习的是贝叶斯个性化排序算法：Bayesian Personalized Ranking。

排序学习设计方法

排序学习的模型通常分为**单点法（Pointwise Approach）**、**配对法（Pairwise Approach）**和**列表法（Listwise Approach）**三大类，三种方法并不是特定的算法，而是排序学习模型的设计思路，主要区别体现在损失函数（Loss Function）、以及相应的标签标注方式和优化方法的不同。

单点法（Pointwise）

单点法排序学习模型的每一个训练样本都仅仅是某一个查询关键字和某一个文档的配对。它们之间是否相关，与其他文档和其他查询关键字都没有关系。很明显，单点法排序学习是对现实的一个极大简化，但是对于训练排序算法来说是一个不错的起点。单点法将文档转换为特征向量后，机器学习系统根据从训练数据中学习到的分类或者回归函数对文档打分，打分结果即是搜索结果。

单点排序学习可以按照标注和损失函数设计的不同，将排序问题转化成回归、分类、和有序分类问题（有些文献也称有序回归）问题，可参考下图：

分别看一下损失函数的设计思想:

- 分类（Classification）：输出空间包含的是无序类别，对每个查询-文档对的样本判断是否相关，可以是二分类的，如相关认为是正例，不相关认为是负例；也可以是类似

NDCG 那样的五级标注的多分类问题。分类模型通常会输出一个概率值，可根据概率值的排序作为排序最终结果。

- 回归（Regression）：输出空间包含的是真实值相关度得分，可通过回归来直接拟合相关度打分。
- 有序分类（Ordinal Classification）：有序分类也称有序回归（Ordinal Regression），输出空间一般包含的是有序类别，通常的做法是找到一个打分函数，然后用一系列阈值对得分进行分割，得到有序类别。

单点法（Pointwise）的应用

回到我们的推荐系统领域，最常用就是二元分类的 Pointwise，比如常见的点击率（CTR）预估问题，之所以用得多，是因为二元分类的 Pointwise 模型的复杂度通常比 Pairwise 和 Listwise 要低，而且可以借助用户的点击反馈自然地完成正负样例的标注，而其他 Pairwise 和 Listwise 的模型标注就没那么容易了。成功地将排序问题转化成分类问题，也就意味着我们机器学习中那些常用的分类方法都可以直接用来解决排序问题，如 LR、GBDT、SVM 等，甚至包括结合深度学习的很多推荐排序模型，都属于这种 Pointwise 的思想范畴。

代表算法有：基于神经网络的排序算法 RankProp、基于感知机的在线排序算法 Prank(Perception Rank)/OAP-BPM 和基于 SVM 的排序算法。推荐系统中使用较多的 Pointwise 方法是 LR、GBDT、SVM、FM 以及结合 DNN 的各种排序算法。

单点法（Pointwise）的缺点

Pointwise 方法通过优化损失函数求解最优的参数，可以看到 Pointwise 方法非常简单，工程上也易实现，但是 Pointwise 也存在很多问题：

- Pointwise 只考虑单个文档同 query 的相关性，没有考虑文档间的关系，然而排序追求的是排序结果，并不要求精确打分，只要有相对打分即可；
- 通过分类只是把不同的文档做了一个简单的区分，同一个类别里的文档则无法深入区别，虽然我们可以根据预测的概率来区别，但实际上，这个概率只是准确度概率，并不是真正的排序靠前的预测概率；
- Pointwise 方法并没有考虑同一个 query 对应的文档间的内部依赖性。一方面，导致输入空间内的样本不是 IID 的，违反了 ML 的基本假设，另一方面，没有充分利用这种样本间的结构性。其次，当不同 query 对应不同数量的文档时，整体 loss 将容易被对应文档数量大的 query 组所支配，应该每组 query 都是等价的才合理。
- 很多时候，排序结果的 Top N 条的顺序重要性远比剩下全部顺序重要性要高，因为损失函数没有相对排序位置信息，这样会使损失函数可能无意的过多强调那些不重要的

docs，即那些排序在后面对用户体验影响小的 doc，所以对于位置靠前但是排序错误的文档应该加大惩罚。

配对法（Pairwise）

配对法的基本思路是对样本进行两两比较，构建偏序文档对，从比较中学习排序，因为对于一个查询关键字来说，最重要的其实不是针对某一个文档的相关性是否估计得准确，而是要能够正确估计一组文档之间的“相对关系”。因此，Pairwise 的训练集样本从每一个“关键字文档对”变成了“关键字文档文档配对”。也就是说，每一个数据样本其实是一个比较关系，当前一个文档比后一个文档相关排序更靠前的话，就是正例，否则便是负例，如下图。试想，有三个文档：A、B 和 C。完美的排序是“B>C>A”。我们希望通过学习两两关系“B>C”、“B>A”和“C>A”来重构“B>C>A”。

这里面有3个非常关键的假设：

- 我们可以针对某一个关键字得到一个完美的排序关系。在实际操作中，这个关系可以通过五级相关标签来获得，也可以通过其他信息获得，比如点击率等信息。然而，这个完美的排序关系并不是永远都存在的。试想在电子商务网站中，对于查询关键字“哈利波特”，有的用户希望购买书籍，有的用户则希望购买含有哈利波特图案的 T 恤，显然，这里面就不存在一个完美排序。
- 我们寄希望能够学习文档之间的两两配对关系从而“重构”这个完美排序。然而，这也不是一个有“保证”的思路。用刚才的例子，希望学习两两关系“B>C”、“B>A”和“C>A”来重构完美排序“B>C>A”。然而，实际中，这三个两两关系之间是独立的。特别是在预测的时候，即使模型能够正确判断“B>C”和“C>A”，也不代表模型就一定能得到“B>A”。注意，这里的关键是“一定”，也就是模型有可能得到也有可能得不到。两两配对关系不能“一定”得到完美排序，这个结论其实就揭示了这种方法的不一致性。也就是说，我们并不能真正保证可以得到最优的排序。
- 我们能够构建样本来描述这样的两两相对的比较关系。一个相对比较简单的情况，认为文档之间的两两关系来自于文档特征（Feature）之间的差异。也就是说，可以利用样本之间特征的差值当做新的特征，从而学习到差值到相关性差异这样的一组对应关系。

Pairwise 最终的算分，分类和回归都可以实现，不过最常用的还是二元分类，如下图：

配对法（Pairwise）的应用

代表算法：基于 SVM 的 Ranking SVM 算法、基于神经网络的 RankNet 算法和基于 Boosting 的 RankBoost 算法。推荐系统中使用较多的 Pairwise 方法是贝叶斯个性化排序（Bayesian personalized ranking，BPR）。

配对法（Pairwise）的缺点

Pairwise 方法通过考虑两两文档之间的相关对顺序来进行排序，相比 Pointwise 方法有明显改善。但 Pairwise 方法仍有如下问题：

- 使用的是两文档之间相关度的损失函数，而它和真正衡量排序效果的指标之间存在很大不同，甚至可能是负相关的，如可能出现 Pairwise Loss 越来越低，但 NDCG 分数也越来越低的现象。
- 只考虑了两个文档的先后顺序，且没有考虑文档在搜索列表中出现的位置，导致最终排序效果并不理想。
- 不同的查询，其相关文档数量差异很大，转换为文档对之后，有的查询可能有几百对文档，有的可能只有几十个，这样不加均一化地在一起学习，模型会优先考虑文档对数量多的查询，减少这些查询的 loss，最终对机器学习的效果评价造成困难。
- Pairwise 方法的训练样例是偏序文档对，它将对文档的排序转化为对不同文档与查询相关性大小关系的预测；因此，如果因某个文档相关性被预测错误，或文档对的两个文档相关性均被预测错误，则会影响与之关联的其它文档，进而引起连锁反应并影响最终排序结果。

列表法（Listwise）

相对于尝试学习每一个样本是否相关或者两个文档的相对比较关系，列表法排序学习的基本思路是尝试直接优化像 NDCG（Normalized Discounted Cumulative Gain）这样的指标，从而能够学习到最佳排序结果。列表法的相关研究有很大一部分来自于微软研究院。列表法排序学习有两种基本思路。第一种称为 Measure-specific，就是直接针对 NDCG 这样的指标进行优化。目的简单明了，用什么做衡量标准，就优化什么目标。第二种称为 Non-measure specific，则是根据一个已经知道的最优排序，尝试重建这个顺序，然后来衡量这中间的差异。

1）Measure-specific，直接针对 NDCG 类的排序指标进行优化

直接优化排序指标的难点在于，希望能够优化 NDCG 指标这样的“理想”很美好，但是现实却很残酷。NDCG、MAP 以及 AUC 这类排序标准，都是在数学的形式上的“非连续”（Non-Continuous）和“非可微分”（Non-Differentiable）。而绝大多数的优化算法都

是基于“连续”（Continuous）和“可微分”（Differentiable）函数的。因此，直接优化难度比较大。

针对这种情况，主要有这么几种解决方法。

- 既然直接优化有难度，那就找一个近似 NDCG 的另外一种指标。而这种替代的指标是“连续”和“可微分”的 。只要我们建立这个替代指标和 NDCG 之间的近似关系，那么就能够通过优化这个替代指标达到逼近优化 NDCG 的目的。这类的代表性算法的有 SoftRank 和 AppRank。
- 尝试从数学的形式上写出一个 NDCG 等指标的“边界”（Bound），然后优化这个边界。比如，如果推导出一个上界，那就可以通过最小化这个上界来优化 NDCG。这类的代表性算法有 SVM-MAP 和 SVM-NDCG。
- 希望从优化算法上下手，看是否能够设计出复杂的优化算法来达到优化 NDCG 等指标的目的。对于这类算法来说，算法要求的目标函数可以是“非连续”和“非可微分”的。这类的代表性算法有 AdaRank 和 RankGP。

2）Non-measure specific，尝试重建最优顺序，衡量其中差异

这种思路的主要假设是，已经知道了针对某个搜索关键字的完美排序，那么怎么通过学习算法来逼近这个完美排序。我们希望缩小预测排序和完美排序之间的差距。值得注意的是，在这种思路的讨论中，优化 NDCG 等排序的指标并不是主要目的。这里面的代表有 ListNet 和 ListMLE。

3）列表法和配对法的中间解法

第三种思路某种程度上说是第一种思路的一个分支，因为很特别，这里单独列出来。其特点是在纯列表法和配对法之间寻求一种中间解法。具体来说，这类思路的核心思想，是从 NDCG 等指标中受到启发，设计出一种替代的目标函数。这一步和刚才介绍的第一种思路中的第一个方向有异曲同工之妙，都是希望能够找到替代品。找到替代品以后，接下来就是把直接优化列表的想法退化成优化某种配对。这第二步就更进一步简化了问题。这个方向的代表方法就是微软发明的 LambdaRank 以及后来的 LambdaMART。微软发明的这个系列算法成了微软的搜索引擎 Bing 的核心算法之一，而且 LambdaMART 也是推荐领域中可能用到一类排序算法。

列表法（Listwise）的应用

代表算法：基于 Measure-specific 的 SoftRank、SVM-MAP、SoftRank、LambdaRank、LambdaMART，基于 Non-measure specific 的 ListNet、ListMLE、BoltzRank。推荐中使用较多的 Listwise 方法是 LambdaMART。

列表法（Listwise）的缺点

列表法相较单点法和配对法针对排序问题的模型设计更加自然，解决了排序应该基于 query 和 position 问题。但列表法也存在一些问题：

一些算法需要基于排列来计算 loss，从而使得训练复杂度较高，如 ListNet 和 BoltzRank。

位置信息并没有在 loss 中得到充分利用，可以考虑在 ListNet 和 ListMLE 的 loss 中引入位置折扣因子。

BPR（贝叶斯个性化排序）简介

显式反馈：用户对物品的评分，如电影评分；隐式反馈：用户对物品的交互行为，如浏览，购买等，现实中绝大部分数据属于隐式反馈，可以从日志中获取。BPR是基于用户的隐式反馈，为用户提供物品的推荐，并且是直接对排序进行优化。贝叶斯个性化排序（Bayesian personalized ranking，BPR）是一种 Pairwise 方法，并且借鉴了矩阵分解的思路。在开始深入讲解原理之前我们先了解整个 BPR 的基础假设以及基本设定。

因为是基于贝叶斯的 Pairwise 方法，BPR 有两个基本假设：

- 每个用户之间的偏好行为相互独立，即用户 u 在商品 i 和 j 之间的偏好和其他用户无关。
- 同一用户对不同物品的偏序相互独立，也就是用户 u 在商品 i 和 j 之间的偏好和其他的商品无关。

定义

U代表所有的用户user集合；I代表所有的物品item集合；S代表所有用户的隐式反馈，。如下图所示，只要用户对某个物品产生过行为，就标记为+, 所有+样本构成了S。那些未观察到的数据（即用户没有产生行为的数据）标记为?。

代表了用户u产生过行为的物品集合

代表了对物品i产生过行为的用户集合

在原始论文中，BPR 用来解决隐式反馈的推荐排序问题，假设有用户集 U 和物品集 I，当用户 u（u∈U）在物品展示页面点击了物品 i（i∈I）却没有点击同样曝光在展示页面的物品 j（j∈I），则说明对于物品 i 和物品 j，用户 u 可能更加偏好物品 i，用 Pairwise 的思想则是物品 i 的排序要比物品 j 的排序更靠前，这个偏序关系可以写成一个三元组 <u,i,j>，为了简化表述，我们用 符号表示用户 u 的偏好，<u,i,j> 可以表示为：。单独用 代表用户 u 对应的所有商品中两两偏序关系，可知 ，且 满足下面的特性：

- 完整性：
- 反对称性：
- 传递性：

传统解决方式

在使用隐式反馈的情况下，我们会发现观察到的数据均为正例（因为用户对物品交互过才会被观察到），而那些没有被观察到的数据（即用户还没有产生行为的物品），分为两种情况，一种是用户确实对该物品没有兴趣（负类），另一种则是缺失值（即用户以后可能会产生行为的物品）。

传统的个性化推荐通常是计算出用户u对物品i的个性化分数\bar{x}_{ui} ,然后根据个性化分数进行排序。为了得到训练数据，通常是将所有观察到的隐式反馈 (u,i) \in S作为正类，其余所有数据作为负类，如下图所示，左图为观察到的数据，右图为填充后的训练数据：

在填零的情况下，我们的优化目标变成了希望在预测时观测到的数据预测为1，其余的均为0. 于是产生的问题是，我们希望模型在以后预测的缺失值，在训练时却都被认为是负类数据。因此，如果这个模型训练的足够好，那么最终得到的结果就是这些未观察的样本最后的预测值都是0。

矩阵分解的一些缺陷

我们知道，矩阵分解是通过预测用户对候选物品的评分，然后根据这个预测评分去排序，最后再推荐给用户。这种方法是一种典型的 Pointwise 方法，无论是预测评分还是预测隐式反馈，本质上都是在预测用户对一个物品的偏好程度。但是这种方法有很大的问题，因为很多时候我们只能收集到少数正例样本，剩下的数据其实是真实负例和缺失值的混合构成（这里的缺失值是指训练数据中除正例和负例外的未知数据，可以理解为未曝光或者曝光了的但是

用户可能没有注意到缺失数据，所以缺失值中的样本即有可能是正例，也有可能是负例），而我们用这种方法构建训练数据的时候，往往无法确定负例到底是哪些，就只能把除正例以外的其他部分都当作是负例，这就会使得训练数据中负例的一部分其实是缺失值。把缺失值当作是负样本，再以预测误差为评判标准去使劲逼近这些样本。逼近正样本没问题，但是同时逼近的负样本只是缺失值而已，真正呈现在用户面前，并不能确定是不喜欢还是喜欢。而且，这样的模型仅能预测正例或负例，对于类别内的样本无法深入区别其重要性，不利于排序。当然，对于这种情况，我们也可以用一些其他方法来规避这些问题，比如负例采样，比如按预测概率排序，但这些方法也仅仅是“缓兵之计”，对于解决排序问题来说并不完善。我们来看看 BPR 是怎么解决的，它是如何采用 Pairwise 方法来重新优化矩阵分解的。

BPR 的样本构建

首先 BPR 利用 Pairwise 的思想来构建偏序关系，它依然没有从无反馈数据中去区分负例样本和缺失值，不过和之前的方法不一样的是，BPR 不是单纯地将无反馈数据都看做是负例，而是与正例结合一起来构建偏序关系。这里的核心假设是，某用户对他有过反馈的物品的偏好程度一定比没有反馈过的物品高（这里的反馈一般指隐式反馈，如点击浏览等，不涉及负反馈），未反馈的物品包括真正的负例以及缺失值。BPR 试图通过用户的反馈矩阵 S 来为每一个用户构建出完整的偏序关系，也称全序关系，用表示。如下图：

BPR（贝叶斯个性化排序）模型

BPR模型本质上是一种矩阵分解算法，所以其模型本质上是为了将用户物品矩阵分解成两个低维矩阵，再由两个低维矩阵相乘后得到完整的矩阵。对于用户集U和物品集I对应的U*I的预测排序矩阵，我们期望得到两个分解后的用户矩阵 和物品矩阵，满足：

$$\bar{X} = WH^T$$

那么对于任意一个用户u，对应的任意一个物品i，我们预测得出的用户对该物品的偏好计算如下：

$$\bar{x} = w_u \cdot h_i = \sum_{f=1}^{k} w_{uf} h_{if}$$

而模型的最终目标是寻找合适的矩阵W和H，让和(实际的评分矩阵)最相似。看到这里，也许你会说，BPR和矩阵分解没有什区别呀？是的，到目前为止的基本思想是一致的，但是具体的算法运算思路，确实千差万别的，我们慢慢道来。

算法运算思路

BPR 基于最大后验估计来求解模型参数W、H，这里我们用来表示参数W和H，代表用户u对应的所有商品的全序关系，则优化目标是。根据贝叶斯公式，我们有：

$$P(\theta|>_u)=\frac{P(>_u|\theta)P(\theta)}{P(>_u)}$$

由于我们求解假设了用户的排序和其他用户无关，那么对于任意一个用户u来说，P(>u)对所有的物品一样，所以有：

$$P(\theta|>_u)\propto P(>_u|\theta)P(\theta)$$

这个优化目标转化为两部分。第一部分和样本数据集D有关，第二部分和样本数据集D无关。

第一部分

对于第一部分，由于我们假设每个用户之间的偏好行为相互独立，同一用户对不同物品的偏序相互独立，所以有：

$$\prod_{u\in U}P(>_u|\theta)=\prod_{(u,i,j)\in(U\times I\times I)}P(i>_u j|\theta)^{\delta((u,i,j)\in D)}(1-P(i>_u j|\theta))^{\delta((u,i,j)\notin D)}$$

其中：

$$\delta(b)=\begin{cases}1 & \text{if b is true}\\ 0 & \text{else}\end{cases}$$

上面的式子类似于极大似然估计，若用户u相比于j来说更偏向i，那么我们就希望出现的概率越大越好。上面的式子可以进一步改写成：

$$\prod_{u \in U} P(>_u | \theta) = \prod_{(u,i,j) \in D} P(i >_u j | \theta)$$

而对于这个概率，我们可以使用下面这个式子来代替：

$$P(i >_u j | \theta) = \sigma(\bar{x}_{uij}(\theta))$$

其中，是sigmoid函数，σ里面的项我们可以理解为用户u对i和j偏好程度的差异，我们当然希望i和j的差异越大越好，这种差异如何体现，最简单的就是差值：

$$\bar{x}_{uij}(\theta) = \bar{x}_{ui}(\theta) - \bar{x}_{uj}(\theta)$$

省略我们可以将式子简略的写为：

$$\bar{x}_{uij} = \bar{x}_{ui} - \bar{x}_{uj}$$

因此优化目标的第一项可以写作：

$$\prod_{u \in U} P(>_u | \theta) = \prod_{(u,i,j) \in D} \sigma(\bar{x}_{ui} - \bar{x}_{uj})$$

显然，对于训练数据中的<u,i,j>，用户更偏好于i，那么我们当然希望在矩阵中对应的值比对应的值大，而且差距越大越好！

第二部分

当的先验分布是正态分布时，其实就是给损失函数加入了正则项，因此我们可以假定的先验分布是正态分布：

$$P(\theta) \sim N(0, \lambda_\theta I)$$

所以：

$$\ln P(\theta) = \lambda \|\theta\|^2$$

因此，最终的最大对数后验估计函数可以写作：

$$\ln P(\theta|>_u) \propto \ln P(>_u|\theta)P(\theta) = \ln \prod_{(u,i,j)\in D} \sigma(\bar{x}_{ui}-\bar{x}_{uj}) + \ln P(\theta) = \sum_{(u,i,j)\in D} \ln \sigma(\bar{x}_{ui}-\bar{x}_{uj}) + \lambda \|\theta\|^2$$

剩下的我们就可以通过梯度上升法(因为是要让上式最大化)来求解了。

算法代码实现

用到的数据集movieslen 100k：https://grouplens.org/datasets/movielens/

数据预处理

首先，我们需要处理一下数据，得到每个用户打分过的电影，同时，还需要得到用户的数量和电影的数量。

```
import numpy as np
import pandas as pd
from collections import defaultdict
import random
import tensorflow as tf


def gen_test(user_ratings):
    """
    对每一个用户u，在user_ratings中随机找到他评分过的一部电影i,保存在user_rati
    后面构造训练集和测试集需要用到。
    """
    user_test = dict()
    for u, i_list in user_ratings.items():
        user_test[u] = random.sample(user_ratings[u], 1)[0]
    return user_test


def gen_train_batch(user_ratings, user_ratings_test, item_list, batch_siz
    """
    构造训练用的三元组
    对于随机抽出的用户u，i可以从user_ratings随机抽出，而j也是从总的电影集中随机
    """
    t = []
    for b in range(batch_size):
        u = random.sample(user_ratings.keys(), 1)[0]
        i = random.sample(user_ratings[u], 1)[0]
        while i == user_ratings_test[u]:
            i = random.sample(user_ratings[u], 1)[0]

        j = random.sample(item_list, 1)[0]
        while j in user_ratings[u]:
            j = random.sample(item_list, 1)[0]
        t.append([u, i, j])
```

```
35      return np.asarray(t)
36
37
38  def gen_test_batch(user_ratings, user_ratings_test, item_list):
39      """
40      对于每个用户u，它的评分电影i是我们在user_ratings_test中随机抽取的，它的j是
41      比如用户u有1000部电影没有评分，那么这里该用户的测试集样本就有1000个
42      """
43      for u in user_ratings.keys():
44          t = []
45          i = user_ratings_test[u]
46          for j in item_list:
47              if not (j in user_ratings[u]):
48                  t.append([u, i, j])
49          yield np.asarray(t)
50
51
52  def bpr_mf(user_count, item_count, hidden_dim):
53      """
54      hidden_dim为矩阵分解的隐含维度k。user_emb_w对应矩阵W, item_emb_w对应矩阵H
55      """
56      u = tf.placeholder(tf.int32, [None])
57      i = tf.placeholder(tf.int32, [None])
58      j = tf.placeholder(tf.int32, [None])
59
60      user_emb_w = tf.get_variable("user_emb_w", [user_count + 1, hidden_di
61                                   initializer=tf.random_normal_initializer
62      item_emb_w = tf.get_variable("item_emb_w", [item_count + 1, hidden_di
63                                   initializer=tf.random_normal_initializer
64
65      u_emb = tf.nn.embedding_lookup(user_emb_w, u)
66      i_emb = tf.nn.embedding_lookup(item_emb_w, i)
67      j_emb = tf.nn.embedding_lookup(item_emb_w, j)
68
69      # MF predict: u_i > u_j
70      # 第一部分的i 和 j的差值计算
71      x = tf.reduce_sum(tf.multiply(u_emb, (i_emb - j_emb)), 1, keep_dims=T
72
73      # AUC for one user:
74      # reasonable iff all (u,i,j) pairs are from the same user
75      # average AUC = mean( auc for each user in test set)
76      mf_auc = tf.reduce_mean(tf.to_float(x > 0))
77
78      # 第二部分的正则项
79      l2_norm = tf.add_n([
80          tf.reduce_sum(tf.multiply(u_emb, u_emb)),
81          tf.reduce_sum(tf.multiply(i_emb, i_emb)),
82          tf.reduce_sum(tf.multiply(j_emb, j_emb))
83      ])
84
85      # 整个loss
86      regulation_rate = 0.0001
87      bprloss = regulation_rate * l2_norm - tf.reduce_mean(tf.log(tf.sigmoi
88
89      # 梯度上升
90      train_op = tf.train.GradientDescentOptimizer(0.01).minimize(bprloss)
91      return u, i, j, mf_auc, bprloss, train_op
92
93
94  if __name__ == "__main__":
95      df = pd.read_csv("ml-100k/u.data", sep='\t', header=None, names=['use
96      user_list = df['user_id'].unique().tolist()
97      item_list = df['item_id'].unique().tolist()
98      user_count = len(user_list)
99      item_count = len(item_list)
```

```
    # print(user_count, item_count)

    user_ratings = defaultdict(set)
    for index, row in df.iterrows():
        u = row['user_id']
        i = row['item_id']
        user_ratings[u].add(i)

    user_ratings_test = gen_test(user_ratings)

    with tf.Session() as sess:
        """
        这里k取了20， 迭代次数3， 主要是为了快速输出结果。
        如果要做一个较好的BPR算法，需要对k值进行选择迭代，并且迭代次数也要更多一
        """
        u, i, j, mf_auc, bprloss, train_op = bpr_mf(user_count, item_coun
        sess.run(tf.global_variables_initializer())

        for epoch in range(1, 4):
            _batch_bprloss = 0
            for k in range(1, 5000): # uniform samples from training set
                uij = gen_train_batch(user_ratings, user_ratings_test, it
                _bprloss, _train_op = sess.run([bprloss, train_op],
                                               feed_dict={u: uij[:, 0], i

                _batch_bprloss += _bprloss

            print("epoch:", epoch)
            print("bprloss:", _batch_bprloss / k)
            print("_train_op")

            user_count = 0
            _auc_sum = 0.0

            for t_uij in gen_test_batch(user_ratings, user_ratings_test,
                _auc, _test_bprloss = sess.run([mf_auc, bprloss],
                                               feed_dict={u: t_uij[:, 0],
                                               )
                user_count += 1
                _auc_sum += _auc
            print("test_loss: ", _test_bprloss, "test_auc: ", _auc_sum /
            print("")
        variable_names = [v.name for v in tf.trainable_variables()]
        values = sess.run(variable_names)
        for k, v in zip(variable_names, values):
            print("Variable: ", k)
            print("Shape: ", v.shape)
            print(v)

    """
    现在已经得到了W,H矩阵，就可以对任意一个用户u的评分排序了。注意输出的W,H矩阵分
    """
    # 0号用户对这个用户对所有电影的预测评分
    session1 = tf.Session()
    u1_dim = tf.expand_dims(values[0][0], 0)
    u1_all = tf.matmul(u1_dim, values[1], transpose_b=True)
    result_1 = session1.run(u1_all)
    print(result_1)

    print("以下是给用户0的推荐：")
    p = np.squeeze(result_1)
    p[np.argsort(p)[:-5]] = 0
    for index in range(len(p)):
        if p[index] != 0:
            print(index, p[index])
```

参考链接：

- https://lumingdong.cn/learning-to-rank-in-recommendation-system.html
- https://www.cnblogs.com/pinard/p/9128682.html
- https://blog.csdn.net/ddydavie/article/details/84331584
- http://media.people.com.cn/n1/2016/0408/c402798-28261483.html
- https://www.jianshu.com/p/ba1936ee0b69
- https://towardsdatascience.com/recommender-system-using-bayesian-personalized-ranking-d30e98bba0b9

打赏作者
微信支付
支付宝

数据

#算法

发表评论

Write a response...

Name

E-mail address

Website Link

发表评论