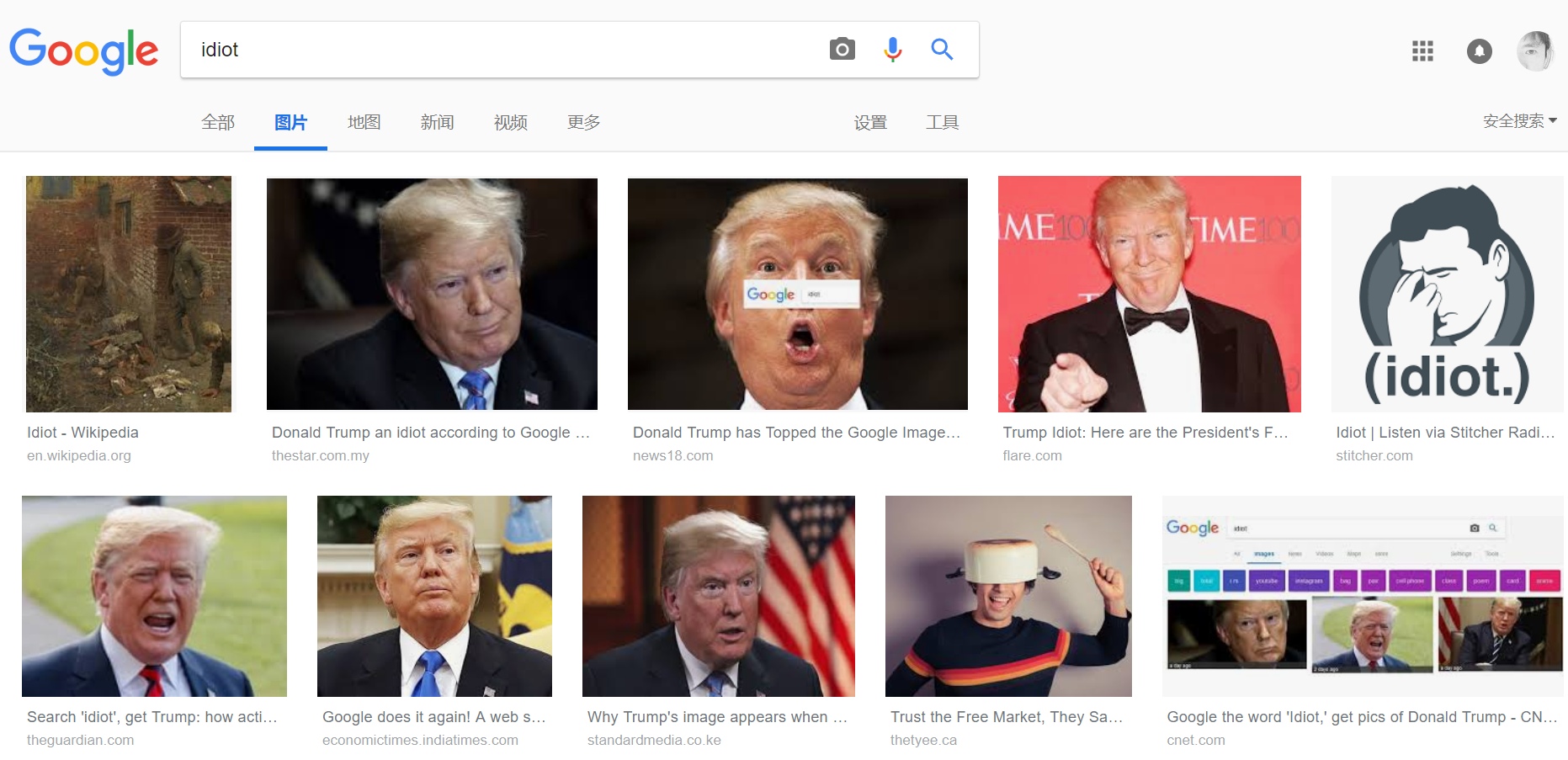
## **1. Learning to Rank**

### **1.1 什么是排序算法**

 为什么google搜索 ”idiot“ 后，会出现特朗普的照片？



 “我们已经爬取和存储了数十亿的网页拷贝在我们相应的索引位置。因此，你输入一个关键字，我们将关键词与网页进行匹配，并根据200多个因子对其进行排名，这些因子包括相关性、新鲜度、流行度、PageRank值、查询和文档匹配的单词个数、网页URL链接地址长度以及其他人对排序结果的满意度等。在此基础上，在任何给定的时间，我们尝试为该查询排序并找到最佳结果。”

　　—— GoogleCEO： 桑达尔·皮查伊

### **1.2 排序算法的发展**

1.2.1 早期排序技术

最早主要是利用词频、逆文档频率和文档长度这几个因子来人工拟合排序公式。因为考虑因素不多，由人工进行公式拟合是完全可行的，此时机器学习并不能派上很大用场，因为机器学习更适合采用很多特征来进行公式拟合。此外，对于有监督机器学习来说，首先需要大量的训练数据，在此基础上才可能自动学习排序模型，单靠人工标注大量的训练数据不太现实。

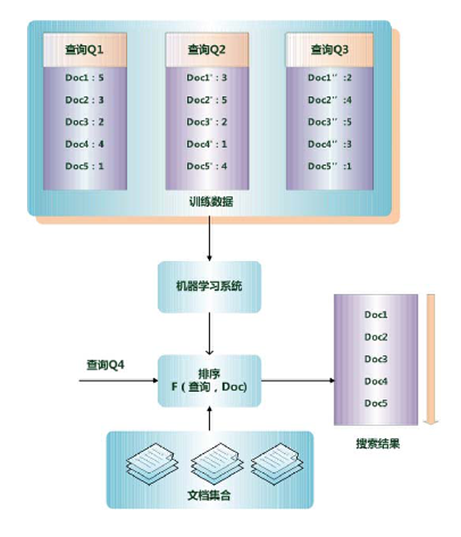
1.2.2 基于机器学习的排序技术

对于搜索引擎来说，尽管无法靠人工来标注大量训练数据，但是用户点击记录是可以当做机器学习方法训练数据的一个替代品，比如用户发出一个查询，搜索引擎返回搜索结果，用户会点击其中某些网页，可以假设用户点击的网页是和用户查询更加相关的页面。

### **1.3 Learning to Rank（LTR）**

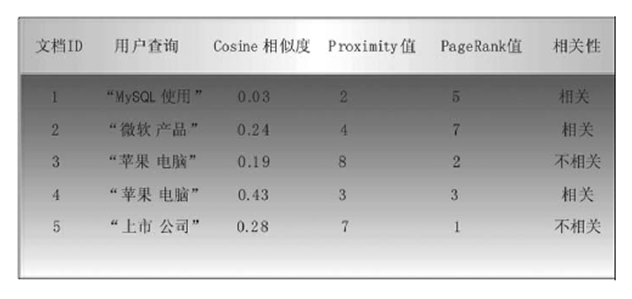
机器学习排序系统由4个步骤组成：

1. 人工标注训练数据
2. 文档特征抽取
3. 学习分类函数
4. 在实际搜索系统中采用机器学习模型



## **2. PointWise Approach**

定义：单文档方法的处理对象是单独的一篇文档，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。



****Score（Q， D）＝a×CS＋b×PM＋c×PR＋d****

 对于某个新的查询Q和文档D，系统首先获得其文档D对应的3个特征的特征值，之后利用学习到的参数组合计算两者得分，当得分大于设定的阈值，即可判断文档是相关文档，否则判断为不相关文档。

## **3. PairWise Approach**

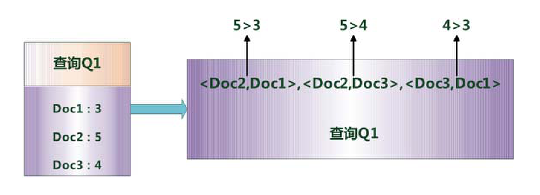
**

对于左、右两张图，按照pointwise的思想，则认为这两条样本 i 和 j 都被点击，因此label都是1。但在右图包含更重要的信息 ：用户只点了红框内的酒店，而没有点黄框内的酒店（右图黄框内的酒店和左图点击红框的酒店一致）。这说明样本 j 的 label应该比样本 i 的label大（样本 j 排名比样本 i 更靠前），很显然，pointwise并没有利用到这个信息。

对于搜索任务来说，系统接收到用户查询后，返回相关文档列表，所以****问题的关键是确定文档之间的先后顺序关系。****

单文档方法（PointWise Approach）完全从单个文档的分类得分角度计算，没有考虑文档之间的顺序关系。

文档对方法（PairWise Approach）则将重点转向了对文档顺序关系是否合理进行判断。之所以被称为文档对方法，是因为这种机器学习方法的训练过程和训练目标，是判断任意两个文档组成的文档对<Doc1,Doc2>是否满足顺序关系，即判断是否Doc1应该排在Doc2的前面。

**

根据转换后的训练实例，就可以利用机器学习方法进行分类函数的学习： 输入一个查询和文档对<Doc1,Doc2>，机器学习排序能够判断这种顺序关系是否成立，如果成立，那么在搜索结果中Doc1应该排在Doc2前面，否则Doc2应该排在Doc1前面。通过这种方式，就完成搜索结果的排序任务。

1. 文档对方法（PairWise Approach）只考虑了两个文档对的相对先后顺序，却没有考虑文档出现在搜索列表中的位置。排在搜索结果前列的文档更为重要，如果前列文档出现判断错误，代价明显高于排在后面的文档。
2. 不同的查询，其相关文档数量差异很大，所以转换为只有十几个对应的文档对，这对机器学习系统的效果评价造成困难。

## **4. ListWise Approach**

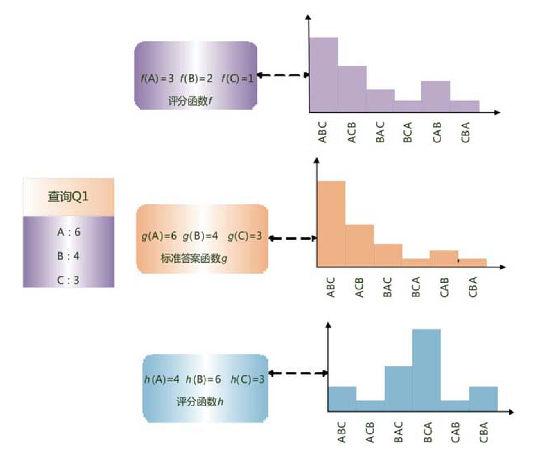
1. 单文档方法（PointWise Approach）将训练集里每一个文档当做一个训练实例。

2. 文档对方法（PairWise Approach）将同一个查询的搜索结果里任意两个文档对作为一个训练实例。

3. 文档列表方法（ListWise Approach）与上述两种表示方式不同，是将每一个查询对应的所有搜索结果列表整体作为一个训练实例，这也是为何称之为文档列表方法的原因。

4. 文档列表方法根据K个训练实例（一个查询及其对应的所有搜索结果评分作为一个实例）训练得到最优评分函数F。对于一个新的用户查询，函数F对每一个文档打分，之后按照得分顺序由高到低排序，就是对应的搜索结果。

对于某个评分函数 f 来说，对3个搜索结果文档的相关性打分，得到3个不同的相关度得分F(A)、 F(B)和F(C)，根据这3个得分就可以计算6种排列组合情况各自的概率值。不同的评分函数，其6种搜索结果排列组合的概率分布是不一样的。所以可以通过不同的评分函数分布与实际分布比较得出最优的那个评分函数作为排序模型。如何判断 h 和 f 与虚拟的最优评分函数 g 更接近？一般可以用两个分布概率之间的距离远近来度量这种相似性，比如 KL散度等。



## **5. Bayesian Personalized Ranking**

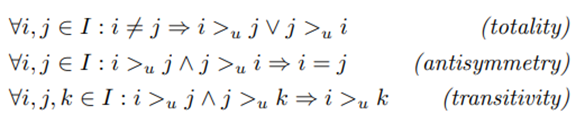
### **5.1 BPR介绍**

* 在推荐系统中，分为召回和排序两个阶段。
* 贝叶斯个性化排序属于Pairwise Approach。

 BPR算法的五个核心知识点：

* 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，即排序关系相互独⽴
* 表⽰⽤户u对 I 的偏好⼤于对 j 的偏好
* 满⾜完整性，反对称性和传递性
* 采用最⼤后验估计计算参数

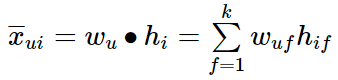
 其中，完整性，反对称性和传递性的定义如下：



### **5.2 BPR参数**

在推荐系统中，排序算法通常完成对候选商品的二次筛选，也叫Reranking。这里的BPR算法借鉴了召回步骤中协同过滤算法的思想： 矩阵分解 。

对于用户u：

****

对于所有用户：

**IMG_264**

其中用户矩阵W：

IMG_265

物品矩阵H：

IMG_266

### **5.3 BPR参数计算方法**

BPR算法采用的是最大化后验概率来估计参数（关于什么是最大化后验概率，可移步我的另外一篇文章：[似然与概率的异同](https://www.cnblogs.com/wkang/p/10069780.html" \t "https://blog.csdn.net/qq_42606051/article/details/_blank)），因此，这里用到了贝叶斯公式。

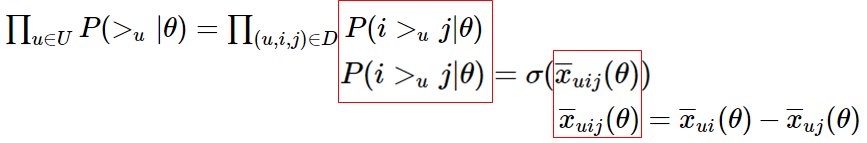
IMG_267

之前已经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：

IMG_268

δ(b)函数返回1 如果条件b成立， 否则返回0。D为训练集， (u,i,j) 表示关系，即相对于j，用户u更喜欢 i 。

由于IMG_269满足完整性和反对称性，所以上式可简化为：



其中，δ()为sigmod函数，用户 u 相比于 j 更喜欢 i 通过借助用户 u 对 i 的喜欢程度与对 j 的喜欢程度的差进行度量。

因此，IMG_271 可表示为：

IMG_272

目标是求解θ。 由于采用最大后验估计来学习参数，所以假设θ服从正态分布：

IMG_273

根据概率密度函数，求得：

IMG_274

关于这个等式的推导，笔者尝试将概率分布带入到概率密度函数中，发现并不能推导出来，但是由于存在正比关系，所以可以近似等于。

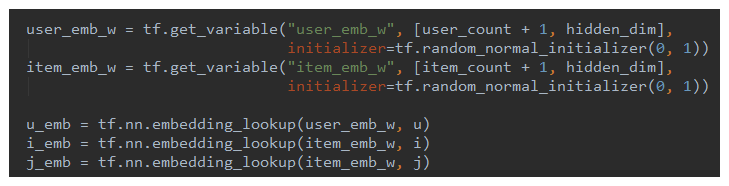
所以，最终的后验概率估计函数为：

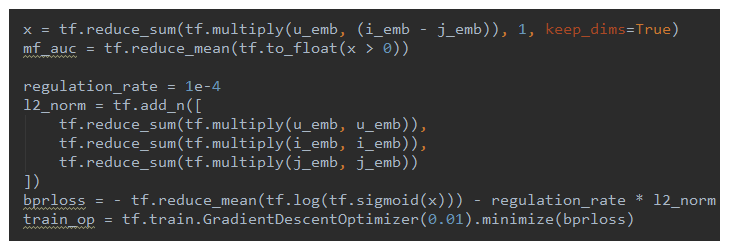
IMG_275

通过最大化这个函数，可以求出参数W和H。

## **6. Bayesian Personalized Ranking算法实现**

网上开源的BPR代码有很多，这里着重表达一下用户embedding矩阵和物品embedding矩阵，以及损失函数的构造。其中损失函数为最小化上一小节的最大后验概率函数。





## **7. 总结**

回顾Bayesian Personalized Ranking 算法，有以下三点值得回味：

1. θ的正态分布（先验）形式：

IMG_278

之所以这样设计，笔者以为有两点：一是方便取对数、二是能与正则化联系起来。

2. 用户 u 相比于 j 更喜欢 i 通过借助用户 u 对 i 的喜欢程度与对 j 的喜欢程度的差进行度量。这当然是最直观的表示方法，当然也可以加以改进。

3. 万物皆可embedding ！通过对用户以及物品分别构造embedding向量，从而完成用户对物品喜好程度的计算。

[https://www.cnblogs.com/wkang/p/10217172.html](https://www.cnblogs.com/wkang/p/10217172.html" \t "https://blog.csdn.net/qq_42606051/article/details/_blank)

# **[贝叶斯个性化排序(BPR)算法小结](https://www.cnblogs.com/pinard/p/9128682.html)**

　　　　在[矩阵分解在协同过滤推荐算法中的应用](https://www.cnblogs.com/pinard/p/6351319.html)中，我们讨论过像funkSVD之类的矩阵分解方法如何用于推荐。今天我们讲另一种在实际产品中用的比较多的推荐算法:贝叶斯个性化排序(Bayesian Personalized Ranking, 以下简称BPR)，它也用到了矩阵分解，但是和funkSVD家族却有很多不同之处。下面我们来详细讨论。

# **1.  BPR算法使用背景**

　　　　在很多推荐场景中，我们都是基于现有的用户和商品之间的一些数据，得到用户对所有商品的评分，选择高分的商品推荐给用户，这是funkSVD之类算法的做法，使用起来也很有效。但是在有些推荐场景中，我们是为了在千万级别的商品中推荐个位数的商品给用户，此时，我们更关心的是用户来说，哪些极少数商品在用户心中有更高的优先级，也就是排序更靠前。也就是说，我们需要一个排序算法，这个算法可以把每个用户对应的所有商品按喜好排序。BPR就是这样的一个我们需要的排序算法。

# **2.  排序推荐算法背景介绍**

　　　　 排序推荐算法历史很悠久，早在做信息检索的各种产品中就已经在使用了。最早的第一类排序算法类别是点对方法(Pointwise Approach)，这类算法将排序问题被转化为分类、回归之类的问题，并使用现有分类、回归等方法进行实现。第二类排序算法是成对方法(Pairwise Approach)，在序列方法中，排序被转化为对序列分类或对序列回归。所谓的pair就是成对的排序，比如(a,b)一组表明a比b排的靠前。我们要讲到的BPR就属于这一类。第三类排序算法是列表方法(Listwise Approach)，它采用更加直接的方法对排序问题进行了处理。它在学习和预测过程中都将排序列表作为一个样本。排序的组结构被保持。

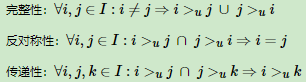
　　　　本文关注BPR，这里我们对排序推荐算法本身不多讲，如果大家感兴趣，可以阅读李航的[A Short Introduction to Learning to Rank](http://times.cs.uiuc.edu/course/598f13/l2r.pdf" \o "A Short Introduction to Learning to Rank" \t "https://www.cnblogs.com/pinard/p/_blank).

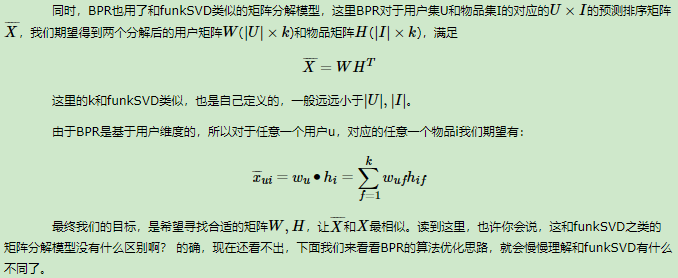
# **3. BPR建模思路**

　　　　在BPR算法中，我们将任意用户u对应的物品进行标记，如果用户u在同时有物品i和j的时候点击了i，那么我们就得到了一个三元组<u,i,j>，它表示对用户u来说，i的排序要比j靠前。如果对于用户u来说我们有m组这样的反馈，那么我们就可以得到m组用户u对应的训练样本。

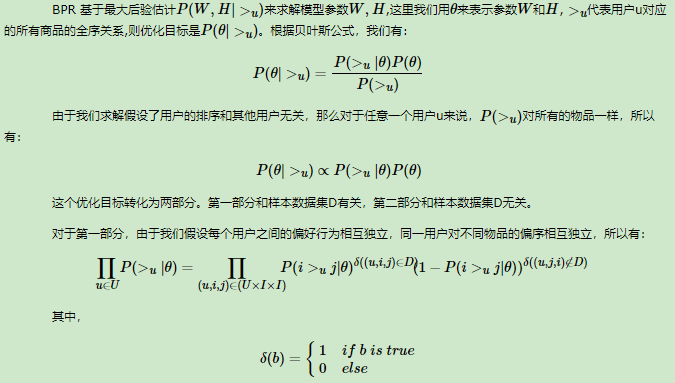
　　　　既然是基于贝叶斯，那么我们也就有假设，这里的假设有两个：一是每个用户之间的偏好行为相互独立，即用户u在商品i和j之间的偏好和其他用户无关。二是同一用户对不同物品的偏序相互独立，也就是用户u在商品i和j之间的偏好和其他的商品无关。为了便于表述，我们用>u>u符号表示用户u的偏好，上面的<u,i,j><u,i,j>可以表示为：i>uj。

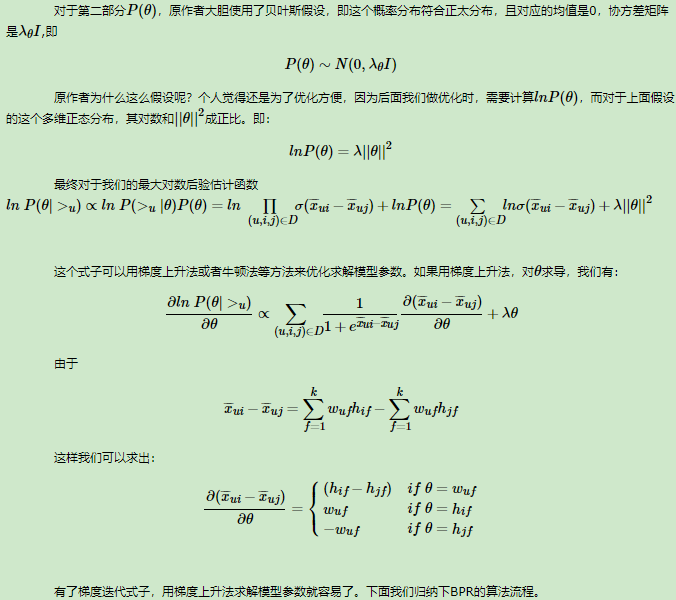
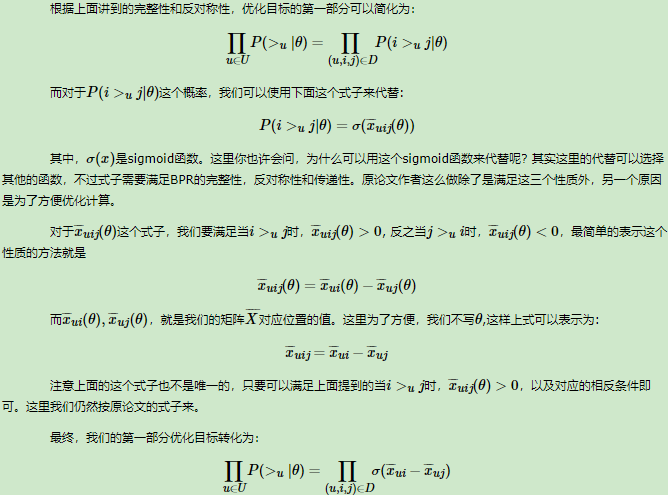
　　　　在BPR中，这个排序关系符号>u满足完全性，反对称性和传递性，即对于用户集U和物品集I：





# **4. BPR的算法优化思路**

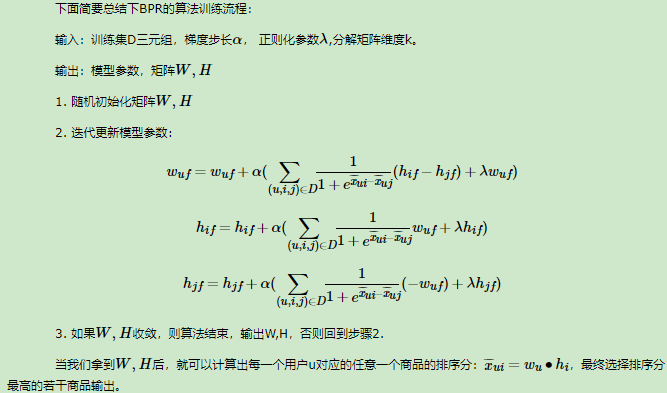




　　　　有了梯度迭代式子，用梯度上升法求解模型参数就容易了。下面我们归纳下BPR的算法流程。

# **5. BPR算法流程**

　　　　下面简要总结下BPR的算法训练流程：



# **6. BPR小结**

　　　　BPR是基于矩阵分解的一种排序算法，但是和funkSVD之类的算法比，它不是做全局的评分优化，而是针对每一个用户自己的商品喜好分贝做排序优化。因此在迭代优化的思路上完全不同。同时对于训练集的要求也是不一样的，funkSVD只需要用户物品对应评分数据二元组做训练集，而BPR则需要用户对商品的喜好排序三元组做训练集。

　　　　在实际产品中，BPR之类的推荐排序在海量数据中选择极少量数据做推荐的时候有优势，因此在某宝某东等大厂中应用也很广泛。由于BPR并不复杂，下一篇我会用tensorflow来做一个BPR的实践，敬请期待。

（欢迎转载，转载请注明出处。欢迎沟通交流： liujianping-ok@163.com）

**评论列表**

[#1楼](https://www.cnblogs.com/pinard/p/9128682.html" \l "4024505) 2018-07-19 22:17 [跳过了E](https://www.cnblogs.com/cdf4745/" \t "https://www.cnblogs.com/pinard/p/_blank) 

刘老师您好！在第4个标题下第三个公式的最后一项“(1−P(i>uj|θ))δ((u,j,i)∉D)”我感觉此处原论文有误，因为按照原论文对D的定义，表达负例应该为“(1−P(i>uj|θ))δ((u,j,i)∈∈D)”或者“(1−P(i>uj|θ))δ((u,i,j)∉D)” 不知我的理解是否正确？

[支持(0)反对(0)](https://www.cnblogs.com/pinard/p/javascript:void(0);)

[#2楼](https://www.cnblogs.com/pinard/p/9128682.html" \l "4024763)[楼主] 2018-07-20 11:31 [刘建平Pinard](https://www.cnblogs.com/pinard/" \t "https://www.cnblogs.com/pinard/p/_blank) 

[@](https://www.cnblogs.com/pinard/p/9128682.html" \l "4024505" \o "查看所回复的评论) 跳过了E  
你好，之前我论文读到这里也是有些纳闷，这里的负例条件的确不好理解。直观上负例应该就是(u,i,j)∉D(u,i,j)∉D,再按完整性和反对称性负例可以加上(u,j,i)∈D(u,j,i)∈D。如果按原文的话(u,j,i)∉D(u,j,i)∉D，也不能把这种条件看做是负例。  
  
由于后面的优化丢弃了负例部分，所以我之前也没有追究。  
  
暂时按你说的修改吧，非常感谢你的指出！

[支持(0)反对(0)](https://www.cnblogs.com/pinard/p/javascript:void(0);)

[#3楼](https://www.cnblogs.com/pinard/p/9128682.html" \l "4048851) 2018-08-23 10:57 [ShawnSu](http://home.cnblogs.com/u/1471533/" \t "https://www.cnblogs.com/pinard/p/_blank) 

刘老师您好！我觉得文章中表达负例的写法是正确的。因为文章中对Ds定义如下：Ds：={(u,i,j)|i∈I+u∩j∈I∖I+u}Ds：={(u,i,j)|i∈Iu+∩j∈I∖Iu+}。举例Figure1中U1而言，就是Ds={(U1,i2,j1),(U1,i2,j4),(u1,i3,j1),(U1,i3,j4)}.根据U1XIXI，它的负例表达为：={(U1,j2,i1),(U1,j2,i4),(u1,j3,i1),(U1,j3,i4)}，并且只有这一种表达。因为（u，i，j）中的i和j都是有限制的。所以1楼的说法应该是有问题的，因为忽略了文章对Ds的定义中i和j的限制条件。所以文章中负例表达成(u,j,i)∉Ds(u,j,i)∉Ds是正确的。不知道理解对否？

[支持(1)反对(0)](https://www.cnblogs.com/pinard/p/javascript:void(0);)

[#4楼](https://www.cnblogs.com/pinard/p/9128682.html" \l "4048939)[楼主] 2018-08-23 11:40 [刘建平Pinard](https://www.cnblogs.com/pinard/" \t "https://www.cnblogs.com/pinard/p/_blank) 

[@](https://www.cnblogs.com/pinard/p/9128682.html" \l "4048851" \o "查看所回复的评论) ShawnSu  
你好，你的理解比较准确，感谢。这里 i,j已经做了限制。感谢指出错误。

[支持(0)反对(0)](https://www.cnblogs.com/pinard/p/javascript:void(0);)

[#5楼](https://www.cnblogs.com/pinard/p/9128682.html" \l "4051260) 2018-08-27 11:01 [巴赫的相对论](http://home.cnblogs.com/u/1472097/" \t "https://www.cnblogs.com/pinard/p/_blank) 

刘老师，您好！  
有一个地方不太明白，为什么在您的这篇文章里  
ln(theta) = lambda\*theta^2,  
而不是  
ln(theta) = -lambda\*theta^2？

[支持(0)反对(0)](https://www.cnblogs.com/pinard/p/javascript:void(0);)

[#6楼](https://www.cnblogs.com/pinard/p/9128682.html" \l "4051506)[楼主] 2018-08-27 14:51 [刘建平Pinard](https://www.cnblogs.com/pinard/" \t "https://www.cnblogs.com/pinard/p/_blank) 

[@](https://www.cnblogs.com/pinard/p/9128682.html" \l "4051260" \o "查看所回复的评论) 巴赫的相对论  
你好，这里λλ是自定义超参数，所以前面是正号或者负号都可以。只要后面保持一致即可。

[支持(0)反对(0)](https://www.cnblogs.com/pinard/p/javascript:void(0);)

[#7楼](https://www.cnblogs.com/pinard/p/9128682.html" \l "4142660) 2018-12-19 11:36 [dasdsadas](http://home.cnblogs.com/u/1566170/" \t "https://www.cnblogs.com/pinard/p/_blank) 

推荐给大家一份超级好用的BPR代码。。。githubcom/RunlongYu/BPR\_MPR

[支持(0)反对(0)](https://www.cnblogs.com/pinard/p/javascript:void(0);)

[#8楼](https://www.cnblogs.com/pinard/p/9128682.html" \l "4193576) 2019-03-04 22:37 [DantinBlack](http://home.cnblogs.com/u/1564817/" \t "https://www.cnblogs.com/pinard/p/_blank) 

刘老师您好！看您的博客有一段时间了，学习到不少知识。我看您似乎是做推荐系统这一块的，不知道您有没有打算开个专栏介绍一些推荐系统相关的知识？除了这些经典的推荐算法外，还有基于社交网络的推荐算法、基于深度学习的推荐算法、实际应用中的推荐系统架构以及一些案例分析？或者您有这方面的较好的学习资源可以推荐一下吗？谢谢！

[支持(0)反对(0)](https://www.cnblogs.com/pinard/p/javascript:void(0);)

[#9楼](https://www.cnblogs.com/pinard/p/9128682.html" \l "4193795)[楼主] 2019-03-05 10:13 [刘建平Pinard](https://www.cnblogs.com/pinard/" \t "https://www.cnblogs.com/pinard/p/_blank) 

[@](https://www.cnblogs.com/pinard/p/9128682.html" \l "4193576" \o "查看所回复的评论) DantinBlack  
你好，后面有这个考虑。现有的比较好的你可以看看刑无刀的推荐系统专栏，还是写的不错的，网上可以找到对应的资料。当然案例分析这个比较具体，可能最好是在项目中自己摸索。

[支持(0)反对(0)](https://www.cnblogs.com/pinard/p/javascript:void(0);)

[#10楼](https://www.cnblogs.com/pinard/p/9128682.html" \l "4194503) 2019-03-05 21:08 [DantinBlack](http://home.cnblogs.com/u/1564817/" \t "https://www.cnblogs.com/pinard/p/_blank) 

[@](https://www.cnblogs.com/pinard/p/9128682.html" \l "4193795" \o "查看所回复的评论) 刘建平Pinard  
好的，十分感谢！

# **[用tensorflow学习贝叶斯个性化排序(BPR)](https://www.cnblogs.com/pinard/p/9163481.html)**

问题：

1. 核心是算法的实现，用tensorflow实现；
2. 输入有哪些，如何保存数据，用到哪些知识点；
3. 如何找到三元组，以及在程序里面如何应用；
4. 理论与代码结合；

https://www.cnblogs.com/pinard/p/9163481.html

　在[贝叶斯个性化排序(BPR)算法小结](https://www.cnblogs.com/pinard/p/9128682.html)中，我们对贝叶斯个性化排序(Bayesian Personalized Ranking, 以下简称BPR)的原理做了讨论，本文我们将从实践的角度来使用BPR做一个简单的推荐。由于现有主流开源类库都没有BPR，同时它又比较简单，因此用tensorflow自己实现一个简单的BPR的算法，下面我们开始吧。

# **1. BPR算法回顾**

　　　　BPR算法是基于矩阵分解的排序算法，它的算法训练集是一个个的三元组<u,i,j><u,i,j>，表示对用户u来说，商品i的优先级要高于商品j。训练成果是两个分解后的矩阵WW和HH,假设有m个用户，n个物品，那么WW的维度是m×km×k, HH的维度是n×kn×k。其中k是一个需要自己定义的较小的维度。对于任意一个用户u，我们可以计算出它对商品i的排序评分为x¯¯¯ui=wu∙hix¯ui=wu∙hi。将u对所有物品的排序评分中找出最大的若干个，就是我们对用户u的真正的推荐集合。

# **2. 基于movieLens 100K做BPR推荐**

　　　　本文我们基于MovieLens 100K的数据做BPR推荐示例，[数据下载链接在这](http://files.grouplens.org/datasets/movielens/ml-100k.zip" \t "https://www.cnblogs.com/pinard/p/_blank)。这个数据集有943个用户对1682部电影的打分。由于BPR是排序算法，因此数据集里的打分会被我们忽略，主要是假设用户看过的电影会比用户满意看的电影的排序评分高。由于tensorflow需要批量梯度下降，因此我们需要自己划分若干批训练集和测试集。

# **3. 算法流程**

　　　　下面我们开始算法的流程，参考了github上一个较旧的BPR代码[于此](https://github.com/dongx-duan/bpr/blob/master/bpr_mf.ipynb" \o "于此" \t "https://www.cnblogs.com/pinard/p/_blank)，有删改和增强。

　　　　完整代码参见我的github:https://github.com/ljpzzz/machinelearning/blob/master/classic-machine-learning/bpr.ipynb

　　　　首先是载入类库和数据，代码如下：

解释：

defaultdict 主要用来需要对 value 做初始化的情形。对于字典来说，key 必须是 hashable，immutable，unique 的数据，而 value 可以是任意的数据类型。如果 value 是 list，dict 等数据类型，在使用之前必须初始化为空，有些情况需要把 value 初始化为特殊值，比如 0 或者 ‘‘。

[?](https://www.jb51.net/article/88147.htm)

|  |  |
| --- | --- |
| 1  2  3  4  5 | from collections import defaultdict    person\_by\_age = defaultdict(list)  for person in persons:    d[person.age].append(person.name) |

defaultdict 和 dict 的使用方法一样，只有在初始化的时候必须传入一个 callable 的对象 x，当访问某个还不存在的 key 时，会把 value 自动设置成 x()。比如上例中，当第一次访问某个年龄的人 d[person.age] 就会变成 list()， 也就是 []。

当然也可以使用自己定义的 callable 对象，比如：

[?](https://www.jb51.net/article/88147.htm)

|  |  |
| --- | --- |
| 1  2  3 | d = defaultdict(lambda: 0)  d["hello"] += 1     # 1  d["a"]         # 0 |

defaultdict 要比 dict.set\_default 效率更高，使用起来也更直观和方便。

标准字典包括一个方法setdefault()来获取一个值，如果值不存在则建立一个默认值。defaultdict初始化容器是会让调用者提前指定默认值。

[?](https://www.jb51.net/article/88147.htm)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | import collections  def default\_factory():    return 'default value'  d = collections.defaultdict(default\_factory, foo = 'bar')  print 'd:', d  print 'foo =>', d['foo']  print 'var =>', d['bar'] |

只要所有键都有相同的默认值，就可以使用这个方法。  
上面的结果是：

[?](https://www.jb51.net/article/88147.htm)

|  |  |
| --- | --- |
| 1  2  3 | d: defaultdict(<function default\_factory at 0x0201FAB0>, {'foo': 'bar'})  foo => bar  var => default value |

import numpy

import tensorflow as tf

import os #Python 的 os 模块封装了常见的文件和目录操作

import random #Python中的random模块用于生成随机数

from collections import defaultdict

#defaultdict 主要用来需要对 value 做初始化的情形。对于字典来说，key 必须是 hashable，

#immutable，unique 的数据，而 value 可以是任意的数据类型。如果 value 是 list，dict 等

#数据类型，在使用之前必须初始化为空，有些情况需要把 value 初始化为特殊值，比如 0 或者 ‘‘。

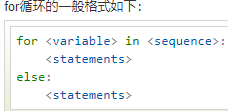
def load\_data(data\_path):

user\_ratings = defaultdict(set)

max\_u\_id = -1

max\_i\_id = -1

with open(data\_path, 'r') as f:# as是打开后的句柄



for line in f.readlines():

u, i, \_, \_ = line.split("\t")

u = int(u)

i = int(i)

user\_ratings[u].add(i)

max\_u\_id = max(u, max\_u\_id)

max\_i\_id = max(i, max\_i\_id)

print ("max\_u\_id:", max\_u\_id)

print ("max\_i\_id:", max\_i\_id)

return max\_u\_id, max\_i\_id, user\_ratings

data\_path = os.path.join('D:\\tmp\\ml-100k', 'u.data')

user\_count, item\_count, user\_ratings = load\_data(data\_path)

　　　　输出为数据集里面的用户数和电影数。同时，每个用户看过的电影都保存在user\_ratings中。

max\_u\_id: 943

max\_i\_id: 1682

　　　　下面我们会对每一个用户u，在user\_ratings中随机找到他评分过的一部电影i,保存在user\_ratings\_test，后面构造训练集和测试集需要用到。

def generate\_test(user\_ratings):

user\_test = dict()

#Python字典是另一种可变容器模型，且可存储任意类型对象，如字符串、数字、元组等其他容器模型。

for u, i\_list in user\_ratings.items():

user\_test[u] = random.sample(user\_ratings[u], 1)[0]

#random.sample()可以从指定的序列中，随机的截取指定长度的片断，不作原地修改。

return user\_test

user\_ratings\_test = generate\_test(user\_ratings)

　　　　接着我们需要得到TensorFlow迭代用的若干批训练集，获取训练集的代码如下，主要是根据user\_ratings找到若干训练用的三元组<u,i,j>，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抽出，当然j必须保证(u,j)不出现在user\_ratings中。

怎么保证j不出现在user\_ratings中？？？

def generate\_train\_batch(user\_ratings, user\_ratings\_test, item\_count, batch\_size=512):

#item\_count项目数

#batch\_size???

t = []

for b in range(batch\_size):

u = random.sample(user\_ratings.keys(), 1)[0]#找一个用户

i = random.sample(user\_ratings[u], 1)[0]#从这个用户评价的分数找一个评分

while i == **user\_ratings\_test[u]**:#当这个评分和测试集里评分一致时

i = random.sample(user\_ratings[u], 1)[0]

j = random.randint(1, item\_count)

while j in user\_ratings[u]:

j = random.randint(1, item\_count)

t.append([u, i, j])

return numpy.asarray(t)

下一步是产生测试集三元组<u,i,j>。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是用户u所有没有评分过的电影集合，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。

def generate\_test\_batch(user\_ratings, user\_ratings\_test, item\_count):

for u in user\_ratings.keys():

t = []

i = user\_ratings\_test[u]

for j in range(1, item\_count+1):

if not (j in user\_ratings[u]):

t.append([u, i, j])

yield numpy.asarray(t)

　　　　有了训练集和测试集，下面是用TensorFlow构建BPR算法的数据流，代码如下，其中hidden\_dim就是我们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。如果大家看过之前写的BPR的算法原理篇，下面的损失函数的构造，相信大家都会很熟悉。

****模型构建****  
首先回忆一下我们需要学习的参数θ，其实就是用户矩阵W(|U|×k)和物品矩阵H(|I|×k)对应的值，对于我们的模型来说，可以简单理解为由id到embedding的转化，因此有：

def bpr\_mf(user\_count, item\_count, hidden\_dim):

u = tf.placeholder(tf.int32, [None])

i = tf.placeholder(tf.int32, [None])

j = tf.placeholder(tf.int32, [None])

with tf.device("/cpu:0"):

user\_emb\_w = tf.get\_variable("user\_emb\_w", [user\_count+1, hidden\_dim],

initializer=tf.random\_normal\_initializer(0, 0.1))

item\_emb\_w = tf.get\_variable("item\_emb\_w", [item\_count+1, hidden\_dim],

initializer=tf.random\_normal\_initializer(0, 0.1))

#embedding\_lookup常用于NLP中将one-hot编码转换为对应的向量编码。

u\_emb = tf.nn.embedding\_lookup(user\_emb\_w, u)

i\_emb = tf.nn.embedding\_lookup(item\_emb\_w, i)

j\_emb = tf.nn.embedding\_lookup(item\_emb\_w, j)

回想一下我们要优化的目标，第一部分是ui和uj对应的预测值的评分之差，再经由sigmoid变换得到的[0,1]值，我们希望这个值越大越好，对于损失来说，当然是越小越好。因此，计算如下：

# MF predict: u\_i > u\_j

x = tf.reduce\_sum(tf.multiply(u\_emb, (i\_emb - j\_emb)), 1, keep\_dims=True)

# tf.multiply（）两个矩阵中对应元素各自相乘

# reduce\_sum应该理解为压缩求和，用于降维

# AUC for one user:

# reasonable iff all (u,i,j) pairs are from the same user

#

# average AUC = mean( auc for each user in test set)

#tf.reduce\_mean 函数用于计算张量tensor沿着指定的数轴（tensor的某一维度）上的

#平均值，主要用作降维或者计算tensor（图像）的平均值。

mf\_auc = tf.reduce\_mean(tf.to\_float(x > 0))

#tf.add\_n([p1, p2, p3....])函数是实现一个列表的元素的相加。

#就是输入的对象是一个列表，列表里的元素可以是向量，矩阵，等

第二部分是我们的正则项，参数就是我们的embedding值，所以正则项计算如下：

l2\_norm = tf.add\_n([

tf.reduce\_sum(tf.multiply(u\_emb, u\_emb)),

tf.reduce\_sum(tf.multiply(i\_emb, i\_emb)),

tf.reduce\_sum(tf.multiply(j\_emb, j\_emb))

])

因此，我们模型整个的优化目标可以写作：

regulation\_rate = 0.0001

bprloss = regulation\_rate \* l2\_norm - tf.reduce\_mean(tf.log(tf.sigmoid(x)))

#tf.train.GradientDescentOptimizer()使用随机梯度下降算法，

#使参数沿着梯度的反方向，即总损失减小的方向移动，实现更新参数

train\_op = tf.train.GradientDescentOptimizer(0.01).minimize(bprloss)

return u, i, j, mf\_auc, bprloss, train\_op

　　　　有了算法的数据流图，训练集和测试集也有了，现在我们可以训练模型求解W,H这两个矩阵了，注意我们在原理篇是最大化对数后验估计函数， 而这里是最小化取了负号后对应的对数后验估计函数，实际是一样的。

代码如下：

with tf.Graph().as\_default(), tf.Session() as session:

u, i, j, mf\_auc, bprloss, train\_op = bpr\_mf(user\_count, item\_count, 20)

session.run(tf.initialize\_all\_variables())

for epoch in range(1, 4):

\_batch\_bprloss = 0

for k in range(1, 5000): # uniform samples from training set

uij = generate\_train\_batch(user\_ratings, user\_ratings\_test, item\_count)

\_bprloss, \_train\_op = session.run([bprloss, train\_op],

feed\_dict={u:uij[:,0], i:uij[:,1], j:uij[:,2]})

\_batch\_bprloss += \_bprloss

print ("epoch: ", epoch)

print ("bpr\_loss: ", \_batch\_bprloss / k)

print ("\_train\_op")

user\_count = 0

\_auc\_sum = 0.0

# each batch will return only one user's auc

for t\_uij in generate\_test\_batch(user\_ratings, user\_ratings\_test, item\_count):

\_auc, \_test\_bprloss = session.run([mf\_auc, bprloss],

feed\_dict={u:t\_uij[:,0], i:t\_uij[:,1], j:t\_uij[:,2]}

)

user\_count += 1

\_auc\_sum += \_auc

print ("test\_loss: ", \_test\_bprloss, "test\_auc: ", \_auc\_sum/user\_count)

print ("")

variable\_names = [v.name for v in tf.trainable\_variables()]

values = session.run(variable\_names)

for k,v in zip(variable\_names, values):

print("Variable: ", k)

print("Shape: ", v.shape)

print(v)

　　　　这里我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些。这里我的输出如下，供参考。

epoch: 1

bpr\_loss: 0.7236263042427249

\_train\_op

test\_loss: 0.76150036 test\_auc: 0.4852939894020929

epoch: 2

bpr\_loss: 0.7229681559433149

\_train\_op

test\_loss: 0.76061743 test\_auc: 0.48528061393838007

epoch: 3

bpr\_loss: 0.7223725006756341

\_train\_op

test\_loss: 0.7597519 test\_auc: 0.4852617720521252

Variable: user\_emb\_w:0

Shape: (944, 20)

[[ 0.08105529 0.04270628 -0.12196594 ... 0.02729403 0.1556453

-0.07148876]

[ 0.0729574 0.01720054 -0.08198593 ... 0.05565814 -0.0372898

0.11935959]

[ 0.03591165 -0.11786834 0.04123168 ... 0.06533947 0.11889934

-0.19697346]

...

[-0.05796075 -0.00695129 0.07784595 ... -0.03869986 0.10723818

0.01293885]

[ 0.13237114 -0.07055715 -0.05505611 ... 0.16433473 0.04535925

0.0701588 ]

[-0.2069717 0.04607181 0.07822093 ... 0.03704183 0.07326393

0.06110878]]

Variable: item\_emb\_w:0

Shape: (1683, 20)

[[ 0.09130769 -0.16516572 0.06490657 ... 0.03657753 -0.02265425

0.1437734 ]

[ 0.02463264 0.13691436 -0.01713235 ... 0.02811887 0.00262074

0.08854961]

[ 0.00643777 0.02678963 0.04300125 ... 0.03529688 -0.11161

0.11927075]

...

[ 0.05260892 -0.03204868 -0.06910443 ... 0.03732759 -0.03459863

-0.05798787]

[-0.07953933 -0.10924194 0.11368059 ... 0.06346208 -0.03269136

-0.03078123]

[ 0.03460099 -0.10591184 -0.1008586 ... -0.07162578 0.00252131

0.06791534]]

　　　　现在我们已经得到了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矩阵分别在values[0]和values[1]中。

　　　　那么我们如何才能对某个用户推荐呢？这里我们以第一个用户为例，它在WW中对应的w0向量为value[0][0]，那么我们很容易求出这个用户对所有电影的预测评分, 代码如下：

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)

　　　　输出为一个评分向量：

[[-0.01707731 0.06217583 -0.01760234 ... 0.067231 0.08989487

-0.05628442]]

　　　　现在给第一个用户推荐5部电影，代码如下：

print("以下是给用户0的推荐：")

p = numpy.squeeze(result\_1)

p[numpy.argsort(p)[:-5]] = 0

for index in range(len(p)):

if p[index] != 0:

print (index, p[index])

　　　　输出如下:

以下是给用户0的推荐：

54 0.1907271

77 0.17746378

828 0.17181025

1043 0.16989286

1113 0.17458326

# **4. 小结**

　　　　以上就是用tensorflow来构建BPR算法模型，并用该算法模型做movieLens 100K推荐的过程。实际做产品项目中，如果要用到BPR算法，一是要注意对隐藏维度k的调参，另外尽量多迭代一些轮数。

　　　　另外我们可以在BPR损失函数那一块做文章。比如我们可以对x¯¯¯uij=x¯¯¯ui−x¯¯¯ujx¯uij=x¯ui−x¯uj这个式子做改进，加上一个基于评分时间的衰减系数，这样我们的排序推荐还可以考虑时间等其他因素。

　　　　以上就是用tensorflow学习BPR的全部内容。

**评论列表**

[#1楼](https://www.cnblogs.com/pinard/p/9163481.html" \l "3998318) 2018-06-14 15:26 [machineworker](http://home.cnblogs.com/u/1412152/" \t "https://www.cnblogs.com/pinard/p/_blank) 

之前一直看你相关的博客，个人也是在广告算法这块做了5-6年时间，最近有个疑惑期望得到你的回复：之前跳槽到一个新的公司做广告相关的算法，后面因为一些原因广告业务做的不好，相关人员都去做推荐相关的工作去了，还有的去做了搜索。感觉搜索这块工程太多了，算法业务也不太一样，我的想法是：现在在工业界落地的算法分支太多，图像识别，搜索推荐广告，深度学习的其它视觉相关的，不可能都去搞一遍吧，是不是应该在一个方面做到十分专业呢（个人觉得广告业务算法比较熟悉），转而去做别的，感觉完全没有优势。这块有什么好的建议吗。

[支持(0)反对(0)](https://www.cnblogs.com/pinard/p/javascript:void(0);)

[#2楼](https://www.cnblogs.com/pinard/p/9163481.html" \l "3998470)[楼主] 2018-06-14 17:23 [刘建平Pinard](https://www.cnblogs.com/pinard/" \t "https://www.cnblogs.com/pinard/p/_blank) 

[@](https://www.cnblogs.com/pinard/p/9163481.html" \l "3998318" \o "查看所回复的评论) machineworker  
你好，如果你在广告业务这一块做的比较多，继续做我觉得挺好的。这一块也有很多值得研究的。基于CTR等的研究现在也很多。  
你可以基于这一块的应用研究相关的机器学习算法，继续深入。既不丢之前的优势，也有提高的空间。

[支持(0)反对(0)](https://www.cnblogs.com/pinard/p/javascript:void(0);)

[#3楼](https://www.cnblogs.com/pinard/p/9163481.html" \l "3998586) 2018-06-14 21:28 [machineworker](http://home.cnblogs.com/u/1412152/" \t "https://www.cnblogs.com/pinard/p/_blank) 

[@](https://www.cnblogs.com/pinard/p/9163481.html" \l "3998470" \o "查看所回复的评论) 刘建平Pinard  
嗯，其实之前在这块做了很多事情，包括架构业务和算法相关的，只是现在由于工作的变动做搜索推荐相关的工作去了，搜索这块感觉工程的东西太多，完全偏离了算法这个方向，可是要继续做广告的话就得重新找份工作（公司业务变更），虽然找份工作不难，关键在目前的公司呆的不久（个人一般至少在一个公司两年才会考虑换个机会），目前想要么继续从事广告算法相关的事情，要么就找一个人脸识别等深度学习相关的公司做一下新的技术。

[支持(0)反对(0)](https://www.cnblogs.com/pinard/p/javascript:void(0);)

[#4楼](https://www.cnblogs.com/pinard/p/9163481.html" \l "3998600)[楼主] 2018-06-14 22:13 [刘建平Pinard](https://www.cnblogs.com/pinard/" \t "https://www.cnblogs.com/pinard/p/_blank) 

[@](https://www.cnblogs.com/pinard/p/9163481.html" \l "3998586" \o "查看所回复的评论) machineworker  
其实做算法和业务都是可以的，不用给自己设个界限。当然如果你不喜欢做工程的东西另说。  
如果不想频繁跳，不妨就尝试下搜索相关的工作，看看能不能适应。同时继续自己在期望的方面的学习。

[支持(0)反对(0)](https://www.cnblogs.com/pinard/p/javascript:void(0);)

[#5楼](https://www.cnblogs.com/pinard/p/9163481.html" \l "3999092) 2018-06-15 14:26 [machineworker](http://home.cnblogs.com/u/1412152/" \t "https://www.cnblogs.com/pinard/p/_blank) 

[@](https://www.cnblogs.com/pinard/p/9163481.html" \l "3998600" \o "查看所回复的评论) 刘建平Pinard  
十分感谢，这块我再思考一下。

[支持(0)反对(0)](https://www.cnblogs.com/pinard/p/javascript:void(0);)

[#6楼](https://www.cnblogs.com/pinard/p/9163481.html" \l "4061729) 2018-09-09 00:19 [乒乓球鸡蛋](https://www.cnblogs.com/crazyant/" \t "https://www.cnblogs.com/pinard/p/_blank) 

请问下楼主，我要做一个feed流推荐；  
我当想到的使用spark mlllib的协同过滤，取出一个user的top k个推荐的物品给用户即可；  
  
可是我发现很多人都在用tf，把feed流推荐作为一个tf的排序问题来求解，是这样吗？问题是tf能分布式计算吗？

[支持(0)反对(0)](https://www.cnblogs.com/pinard/p/javascript:void(0);)

[#7楼](https://www.cnblogs.com/pinard/p/9163481.html" \l "4061903)[楼主] 2018-09-09 13:35 [刘建平Pinard](https://www.cnblogs.com/pinard/" \t "https://www.cnblogs.com/pinard/p/_blank) 

[@](https://www.cnblogs.com/pinard/p/9163481.html" \l "4061729" \o "查看所回复的评论) 乒乓球鸡蛋  
你好，用MLlib的协调过滤是可以的，本质就是一个普通的矩阵分解的过程。  
[https://www.cnblogs.com/pinard/p/6351319.html](https://www.cnblogs.com/pinard/p/6351319.html" \t "https://www.cnblogs.com/pinard/p/_blank)  
但是这个方法本质不是一个排序算法，所以如果对应的场景是排序，找topK，那么BPR一般比普通的矩阵分解要好用。  
  
用TF做feed流推荐，用排序推荐算法现在是很流行的。TF原生就是支持分布式计算的，它的分布式是通过GPU集群并行计算完成的，这点你不用担心。

[支持(0)反对(0)](https://www.cnblogs.com/pinard/p/javascript:void(0);)

[#8楼](https://www.cnblogs.com/pinard/p/9163481.html" \l "4061957) 2018-09-09 15:32 [乒乓球鸡蛋](https://www.cnblogs.com/crazyant/" \t "https://www.cnblogs.com/pinard/p/_blank) 

非常感谢回答，有种又发现一个新领域的感觉  
  
我们团队最近在搭建一个推荐系统，需要满足两个场景：  
1、信息流/猜你喜欢，即针对用户的个性化推荐排序；  
2、相似物品/同购物品/同浏览物品，即物品相似推荐；  
  
对于第1个，看来TF的BPR最合适；  
对于第2个，我理解用spark-mllib的fp-growth算法比较合适，或者用TF做矩阵分解，使用item-factors矩阵的相互相似度getNearestNeibor也可以实现，请问这里有其他的做法吗？

[支持(0)反对(0)](https://www.cnblogs.com/pinard/p/javascript:void(0);)

[#9楼](https://www.cnblogs.com/pinard/p/9163481.html" \l "4062536)[楼主] 2018-09-10 10:53 [刘建平Pinard](https://www.cnblogs.com/pinard/" \t "https://www.cnblogs.com/pinard/p/_blank) 

[@](https://www.cnblogs.com/pinard/p/9163481.html" \l "4061957" \o "查看所回复的评论) 乒乓球鸡蛋  
你好！  
算法没有最合适，只有更合适，试了才知道，毕竟还与你已经拥有的数据有关。  
  
1. 个性化推荐排序的确BPR是很合适的，也可以改造优化，比如加入时间远近的因子等。比如：

x¯¯¯uij=x¯¯¯ui−x¯¯¯uj1timeNow−timeDatax¯uij=x¯ui−x¯uj1timeNow−timeData

2. 相似物品推荐，用关联算法的确很不错，尤其是你数据量到千万级以上的时候。此外，如果你们有做物品画像的话，也可以用物品画像得到的抽象特征，onehot编码后做余弦相似度计算，也是很简单好用的方法。

[支持(0)反对(0)](https://www.cnblogs.com/pinard/p/javascript:void(0);)

[#10楼](https://www.cnblogs.com/pinard/p/9163481.html" \l "4062646) 2018-09-10 12:10 [乒乓球鸡蛋](https://www.cnblogs.com/crazyant/" \t "https://www.cnblogs.com/pinard/p/_blank) 

[@](https://www.cnblogs.com/pinard/p/9163481.html" \l "4062536" \o "查看所回复的评论) 刘建平Pinard  
  
再次感谢！  
  
还有个问题请教  
  
我可用的数据包括：  
用户画像（性别、年龄等人口统计属性）  
物品基本属性（分类、标签、标题、简介等属性）  
用户行为（浏览、点击、收藏、购买等隐式反馈，但是没有评分数据显示反馈）  
  
请问您的这个BRP算法，我能把以上这些都作为特征输入进去吗？  
能做这样的改造吗？

[支持(0)反对(0)](https://www.cnblogs.com/pinard/p/javascript:void(0);)

[#11楼](https://www.cnblogs.com/pinard/p/9163481.html" \l "4063954)[楼主] 2018-09-11 17:42 [刘建平Pinard](https://www.cnblogs.com/pinard/" \t "https://www.cnblogs.com/pinard/p/_blank) 

[@](https://www.cnblogs.com/pinard/p/9163481.html" \l "4062646" \o "查看所回复的评论) 乒乓球鸡蛋  
你好，BPR本身不太好直接放这些特征进去。毕竟它的本质是还是一个矩阵分解的排序算法。  
  
要放也可以，主要是在x¯¯¯uij=x¯¯¯ui−x¯¯¯ujx¯uij=x¯ui−x¯uj这个式子来做文章。比如9楼我讲到的时间因子的用法，其他的特征类似，需要自己去处理转化，不像一般的分类回归算法，可以方便的使用特征。

[支持(0)反对(0)](https://www.cnblogs.com/pinard/p/javascript:void(0);)

[#12楼](https://www.cnblogs.com/pinard/p/9163481.html" \l "4064001) 2018-09-11 18:22 [乒乓球鸡蛋](https://www.cnblogs.com/crazyant/" \t "https://www.cnblogs.com/pinard/p/_blank) 

[@](https://www.cnblogs.com/pinard/p/9163481.html" \l "4063954" \o "查看所回复的评论) 刘建平Pinard  
谢谢  
  
我突然明白了一个重要的关键词“它的本质是还是一个矩阵分解的排序算法”，说明BPR是一个排序算法；  
  
在刑无刀的推荐算法专栏中，我看到这句话：“在推荐系统中，我之前一再强调有挖掘、召回和排序三个阶段”  
  
BPR、CTR、LTR，甚至LR，都是排序阶段可以使用的技术；  
  
而协同过滤、矩阵分解、内容相似、标签推荐这些，都可以用于召回阶段生成候选集，虽然召回阶段的候选集也有相似度顺序，但是因为需要融合所以不会使用；  
  
所以我的画像、用户属性、物品属性这些数据，可以用于召回阶段；

[支持(1)反对(0)](https://www.cnblogs.com/pinard/p/javascript:void(0);)

[#13楼](https://www.cnblogs.com/pinard/p/9163481.html" \l "4151804) 2018-12-29 16:29 [因为专注。所以专业](https://www.cnblogs.com/RHadoop-Hive/" \t "https://www.cnblogs.com/pinard/p/_blank) 

推荐系统搭建，求指导。

[支持(0)反对(0)](https://www.cnblogs.com/pinard/p/javascript:void(0);)

[#14楼](https://www.cnblogs.com/pinard/p/9163481.html" \l "4153445)[楼主] 2019-01-02 10:30 [刘建平Pinard](https://www.cnblogs.com/pinard/" \t "https://www.cnblogs.com/pinard/p/_blank) 

[@](https://www.cnblogs.com/pinard/p/9163481.html" \l "4151804" \o "查看所回复的评论) 因为专注。所以专业  
你好，你这个问题太大了，要看你的问题数据规模，要解决的问题才能评估的，没有普适的方法。

[支持(0)反对(0)](https://www.cnblogs.com/pinard/p/javascript:void(0);)

[#15楼](https://www.cnblogs.com/pinard/p/9163481.html" \l "4165018) 2019-01-17 14:14 [手写的惭愧](http://home.cnblogs.com/u/1556812/" \t "https://www.cnblogs.com/pinard/p/_blank) 

你好楼主，我想请教一下，怎么可以并行呢，我设置了config，但发现没有并行运算

[支持(0)反对(0)](https://www.cnblogs.com/pinard/p/javascript:void(0);)

[#16楼](https://www.cnblogs.com/pinard/p/9163481.html" \l "4165652)[楼主] 2019-01-18 10:32 [刘建平Pinard](https://www.cnblogs.com/pinard/" \t "https://www.cnblogs.com/pinard/p/_blank) 

[@](https://www.cnblogs.com/pinard/p/9163481.html" \l "4165018" \o "查看所回复的评论) 手写的惭愧  
你好，我上面这个程序是单机版的。如果你想多线程，可以使用tf.train.Coordinator来搞。如果想多机器并发，可以使用TensorFlow的ps架构来做，官网有很多资料的。

[支持(0)反对(0)](https://www.cnblogs.com/pinard/p/javascript:void(0);)

BPR的代码

# Implement BPR.

# Steffen Rendle, et al. BPR: Bayesian personalized ranking from implicit feedback.

# Proceedings of the twenty-fifth conference on uncertainty in artificial intelligence. AUAI, 2009.

# @author Runlong Yu, Mingyue Cheng, Weibo Gao

import random

from collections import defaultdict

import numpy as np

from sklearn.metrics import roc\_auc\_score

import scores

class BPR:

user\_count = 943

item\_count = 1682

latent\_factors = 20

lr = 0.01

reg = 0.01

train\_count = 1000

train\_data\_path = 'train.txt'

test\_data\_path = 'test.txt'

size\_u\_i = user\_count \* item\_count

# latent\_factors of U & V

U = np.random.rand(user\_count, latent\_factors) \* 0.01

V = np.random.rand(item\_count, latent\_factors) \* 0.01

biasV = np.random.rand(item\_count) \* 0.01

test\_data = np.zeros((user\_count, item\_count))

test = np.zeros(size\_u\_i)

predict\_ = np.zeros(size\_u\_i)

def load\_data(self, path):

user\_ratings = defaultdict(set)

max\_u\_id = -1

max\_i\_id = -1

with open(path, 'r') as f:

for line in f.readlines():

u, i = line.split(" ") #第一部分内容赋值给u,第二部分内容赋值给i

u = int(u)

i = int(i)

user\_ratings[u].add(i)

max\_u\_id = max(u, max\_u\_id)

max\_i\_id = max(i, max\_i\_id)

return user\_ratings

def load\_test\_data(self, path):

file = open(path, 'r')

for line in file:

line = line.split(' ')

user = int(line[0])

item = int(line[1])

self.test\_data[user - 1][item - 1] = 1

def train(self, user\_ratings\_train):

for user in range(self.user\_count):

# sample a user

u = random.randint(1, self.user\_count)

if u not in user\_ratings\_train.keys():

continue

# sample a positive item from the observed items

i = random.sample(user\_ratings\_train[u], 1)[0]

# sample a negative item from the unobserved items

j = random.randint(1, self.item\_count)

while j in user\_ratings\_train[u]:

j = random.randint(1, self.item\_count)

u -= 1

i -= 1

j -= 1

r\_ui = np.dot(self.U[u], self.V[i].T) + self.biasV[i]

r\_uj = np.dot(self.U[u], self.V[j].T) + self.biasV[j]

r\_uij = r\_ui - r\_uj

loss\_func = -1.0 / (1 + np.exp(r\_uij))

# update U and V

self.U[u] += -self.lr \* (loss\_func \* (self.V[i] - self.V[j]) + self.reg \* self.U[u])

self.V[i] += -self.lr \* (loss\_func \* self.U[u] + self.reg \* self.V[i])

self.V[j] += -self.lr \* (loss\_func \* (-self.U[u]) + self.reg \* self.V[j])

# update biasV

self.biasV[i] += -self.lr \* (loss\_func + self.reg \* self.biasV[i])

self.biasV[j] += -self.lr \* (-loss\_func + self.reg \* self.biasV[j])

def predict(self, user, item):

predict = np.mat(user) \* np.mat(item.T)

return predict

def main(self):

user\_ratings\_train = self.load\_data(self.train\_data\_path)

self.load\_test\_data(self.test\_data\_path)

for u in range(self.user\_count):

for item in range(self.item\_count):

if int(self.test\_data[u][item]) == 1:

self.test[u \* self.item\_count + item] = 1

else:

self.test[u \* self.item\_count + item] = 0

# training

for i in range(self.train\_count):

self.train(user\_ratings\_train)

predict\_matrix = self.predict(self.U, self.V)

# prediction

self.predict\_ = predict\_matrix.getA().reshape(-1)

self.predict\_ = pre\_handel(user\_ratings\_train, self.predict\_, self.item\_count)

auc\_score = roc\_auc\_score(self.test, self.predict\_)

print('AUC:', auc\_score)

# Top-K evaluation

str(scores.topK\_scores(self.test, self.predict\_, 5, self.user\_count, self.item\_count))

def pre\_handel(set, predict, item\_count):

# Ensure the recommendation cannot be positive items in the training set.

for u in set.keys():

for j in set[u]:

predict[(u - 1) \* item\_count + j - 1] = 0

return predict

if \_\_name\_\_ == '\_\_main\_\_':

bpr = BPR()

bpr.main()

[郑州无痛人流哪好](http://www.zztjrlyy.com/" \t "https://blog.csdn.net/qq_42606051/article/details/_blank)