Computer Science and Application 计算机科学与应用, 2021, 11(3), 596-603
Published Online March 2021 in Hans. http://www.hanspub.org/journal/csa
https://doi.org/10.12677/csa.2021.113061

Hans汉斯

基于增量矩阵分解的推荐系统研究

郭思宇，方　睿

成都信息工程大学，计算机学院，四川 成都
Email: 479685283@qq.com, fangrui@cuit.edu.cn

收稿日期：2021年2月20日；录用日期：2021年3月15日；发布日期：2021年3月23日

摘　要

矩阵分解(Matrix Factorization, MF)属于推荐系统中的一种应用广泛且较为经典的算法。它是基于用户行为的推荐算法，被广泛知道的是矩阵分解算法是一种很好的推荐算法。在Netflix Prize算法竞赛中取得了巨大成就，有很多应用矩阵分解拿到冠军的团队，矩阵分解可以有效地解决电影评分预测问题。传统的推荐模型通常采用离线训练的方法，计算出所有训练数据的预测系数用户。在实践中场景中总有一些新用户在训练集中找不到。但我们有他的历史行为记录。那我们对该用户如何进行预测评分呢？最简单的方法是将新用户的数据与旧数据集相结合，再一次进行矩阵分解操作。但是这样的操作计算成本太高了，对于时间成本的要求过高，是行不通的。对于新用户的电影评分预测如何解决这一问题，经过对比分析实验，发现增量矩阵分解算法就可以很好的帮助我们解决新用户电影评分预测的问题，本文将具体阐述如何通过增量更新的方式，大幅度的减少针对新用户电影评分预测的时间，有效的解决新用户进入系统后数据快速更新的问题。每当有新的数据产生，新的用户产品的时候，系统可以快速的对其进行训练，使得模型能够实时应用，优化用户体验。

关键词

矩阵分解，推荐系统，评分预测，增量更新

Research on Recommendation System Based on Incremental Matrix Decomposition

Siyu Guo, Rui Fang

School of Computer Science, Chengdu University of Information Technology, Chengdu Sichuan
Email: 479685283@qq.com, fangrui@cuit.edu.cn

Received: Feb. 20th, 2021; accepted: Mar. 15th, 2021; published: Mar. 23rd, 2021

文章引用: 郭思宇, 方睿. 基于增量矩阵分解的推荐系统研究[J]. 计算机科学与应用, 2021, 11(3): 596-603.
DOI: 10.12677/csa.2021.113061

Abstract

Matrix Factorization (MF) is a widely used and classic algorithm in recommender system. It is a recommendation algorithm based on user behavior. It is widely known that matrix factorization algorithm is a good recommendation algorithm. In the Netflix prize algorithm competition, it has made great achievements, there are many applications of matrix factorization to win the championship team; matrix factorization can effectively solve the problem of film score prediction. Traditional recommendation models usually use off-line training method to calculate the prediction coefficients of all training data. In practice, there are always some new users in the scene that can't be found in the training set. But we have a record of his historical behavior. How do we rate the user? The simplest method is to combine the new user's data with the old data set and perform matrix decomposition again. But the cost of calculation is too high, and the time requirement is too high. For the new user's movie score prediction how to solve this problem, after comparative analysis and experiments, it is found that the incremental matrix decomposition algorithm can help us solve the problem of new user's movie score prediction. This paper will specifically elaborate how to reduce the time of new user's movie score prediction by means of incremental update, and effectively solve the problem of new user's entering the system, the problem of data fast updating after the unification. Whenever there are new data and new user products, the system can train them quickly, so that the model can be applied in real time and the user experience can be optimized.

Keywords

Matrix Decomposition, Recommendation System, Score Prediction, Incremental Update

1. 引言

如今时代经济发展迅速，对于消费的定义也发生着变化，消费不断进行着升级，用户需求个性化已经形成一种趋势[1]。推荐系统可以有效的捕获用户对于关注点的个性化偏好，同时更是可以有效的提升用户满意度，平台的收入也处于大幅上升趋势[2]。以此为主因开始关注和使用推荐系统的平台也占着大比重。推荐系统对用户的历史显性数据及隐性数据进行分析[3]，为用户推荐定制化口味的产品。一个受大家认可的个性化推荐可以用户体验大幅度增强，电子商务巨头亚马逊和 Netflix 已经将推荐系统作为其网站的重要组成部分。推荐系统对娱乐产品，如电影、音乐和电视节目有着特别显著的效果[4]。每个用户电影口味是不同的，越来越多的客户喜欢自己口味的电影[5]。事实证明，用户愿意表明他们对特定电影的满意程度，因此可以获得大量关于哪些电影吸引哪些客户的数据[6]。公司可以通过分析这些数据为用户推荐符合口味的电影。

推荐系统要做的事情就是为每个用户找到其感兴趣的物品推荐给他，做到千人千面。在各种推荐算法中，矩阵分解(Matrix Factorization, MF)是一种经典且广泛应用的算法[7]，在基于用户行为的推荐算法里，矩阵分解算法有着很好的推荐效果，它通过潜在因子的向量来表征用户和项目[8]。矩阵分解算法是基于用户历史行为数据的算法，通过用户的历史行为数据，为用户推荐可能感兴趣的物品，预测其对于该物品的评分。

矩阵分解可以十分有效地解决电影评分预测的问题，传统的推荐模型一般是将训练集的数据采用离线的训练方式，全量地进行导入训练，计算出所有用户的预测评分[9]。在实际的场景中，往往会不断有新的用户出现，在训练集里没出现过，但是我们有他的历史行为数据，那我们如何计算该用户的预测评分呢？最简单的方式就是将这个新用户的数据合并到旧的数据集中，重新做一次矩阵分解[10] [11]，但是这样子的计算代价太大，在时间上是不可行的[12]。本文为了解决新用户的电影评分预测问题，根据用户历史行为数据，先固定物品向量，在不重算所有用户向量的前提下，快速计算出新用户向量，这类模型成为增量模型[13]，因此本文提出的模型称为增量矩阵分解模型。

2. 矩阵分解

矩阵分解，就是把用户和物品都映射到一个 K 维空间上，这个 k 维空间不是直接看到的，通常称为隐因子。每一个物品都得到一个向量 q，每一个用户也得到一个向量 p。对于物品，与它对应的向量 q 中的元素，有正有负，代表着这个物品背后暗藏的一些用户关注的因素，对于用户，与它对应的向量 p 中的元素，也有正有负，代表这个用户在若干因素上的偏好。物品被关注的因素和用户偏好的因素，它们的数量和意义是一致的，就是我们在矩阵分解之处人为指定的 k。

矩阵分解算法是基于用户行为的算法，为此需要先获取用户的历史行为数据，然后以此为依据去预测用户可能感兴趣的物品[14]。将收集到的用户行为数据通过矩阵的方式展示出来，矩阵分解算法要做的事情就是去预测出矩阵中所有空白处的评分，并且使得预测评分的大小能反映用户喜欢的程度，预测评分越大表示用于越可能喜欢。这样我们就可以把预测评分最高的前 K 个物品推荐给用户。

简单来说，矩阵分解算法是把评分矩阵分解为两个矩阵乘积的算法，如图 1：

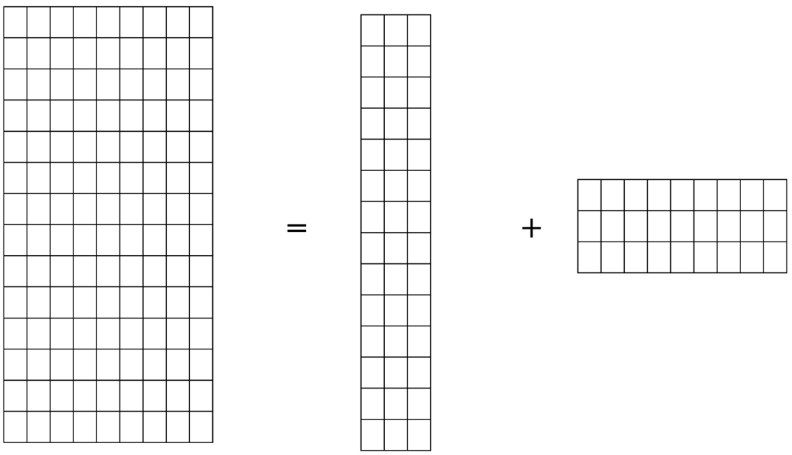

Figure 1. Graphic of matrix decomposition algorithm
图 1. 矩阵分解算法图解

在图中，等号的左边是评分矩阵，是已知数据，它被矩阵分解算法分解为等号右边的两个矩阵的乘积，其中一个被称为用户矩阵，另一个被称为物品矩阵。如果评分矩阵有 n 行 m 列(即 n 个用户，m 个物品)，那么分解出来的用户矩阵就会有 n 行 k 列，其中，第 i 行构成的向量用于表示第 i 个用户。物品矩阵则有 k 行 m 列，其中，第 j 列构成的向量用于表示第 j 个物品。这里的 k 是一个远小于 n 和 m 的正整数。当我们要计算第 i 个用户对第 j 个物品的预测评分时，我们就可以用用户矩阵的第 i 行和物品矩阵的第 j 列做内积，这个内积的值就是预测评分了。

那矩阵分解是如何从评分矩阵中分解出用户矩阵和物品矩阵的呢？简单来说，矩阵分解把用户矩阵和物品矩阵作为未知量，用它们表示出每个用户对每个物品的预测评分，然后通过最小化预测评分跟实际评分的差异，学习出用户矩阵和物品矩阵。也就是说，图中只有等号左边的矩阵是已知的，等号右边的用户矩阵和物品矩阵都是未知量，由矩阵分解通过最小化预测评分跟实际评分的差异学出来的。评分矩阵可以被矩阵算法分解为如下用户矩阵和物品矩阵的乘积，如下表 1。

Table 1. User Item matrix examples
表 1. User Item 矩阵实例

User 矩阵			
	恐怖	剧情	动作
用户 1	0.996798	0.38562772	1.1989608
用户 2	−1.2222522	1.3462118	−0.13543855
用户 3	2.0693302	0.15576078	0.14011484
用户 4	1.2987686	0.54423344	1.7259812
用户 5	0.82122266	0.15917483	1.3019549
用户 6	0.95715284	0.7609615	1.2366726
用户 7	0.800499	1.4115356	0.31377858
用户 8	0.8802036	0.18794502	0.23110229
用户 9	0.96255916	0.3179829	1.080837
Item 矩阵			
	出生入死	最终判决	假日情谊	夏季之白
恐怖	1.514824	0.7324305	0.80080956	0.42376563
剧情	0.71127015	0.7247593	0.8379608	0.8447053
动作	0.7630977	1.0970819	0.5928742	0.8099476

因为用户向量和物品向量的内积等于预测评分，所以在我们学出用户矩阵和物品矩阵之后，把这两个矩阵相乘，就能得到每个用户对每个物品的预测评分了，评分值越大，表示用户喜欢该物品的可能性越大，该物品就越值得推荐给用户。下面的表 2 给出了表 1 中的用户矩阵和物品矩阵相乘的结果，我们把它称为预测评分矩阵，如下表 2。

Table 2. Prediction scoring matrix
表 2. 预测评分矩阵

	出生入死	最终判决	假日情谊	夏季之白
用户 1	6.7394266	5.640459	5.5524774	2.9836497
用户 2	6.1560307	4.85246	5.750779	1.8784995
用户 3	8.823769	5.6828885	6.458826	2.8309512
用户 4	5.6513176	5.0451264	4.848589	2.4187038
用户 5	8.564197	6.746207	7.1328883	3.6082907
用户 6	6.3516283	4.502749	5.6876373	3.8123322
用户 7	5.697568	3.98661	4.98732	3.5189338
用户 8	5.7896385	4.461133	5.3040524	4.048918
用户 9	5.7319975	4.8657384	5.9837894	4.261621

矩阵分解算法的输入是用户对物品的评分矩阵，输出是用户矩阵和物品矩阵[15]，其中用户矩阵的每一行代表一个用户向量，物品矩阵的每一列代表一个物品的向量。用户对物品的预测评分用它们的向量内积来表示，通过最小化预测评分和实际评分的差异来学习用户矩阵和物品矩阵。用矩阵表示即为：

$$\tilde{R} = QP^T$$

事实上用户的行为数据是一个非常非常稀疏的矩阵[16]，因为大部分用户只看过全部电影中很少一部分。如何利用这个矩阵去找潜在因子呢？本文主要应用到的是矩阵的 UV 分解。也就是将上面的评分矩阵分解为两个低维度的矩阵，用 Q 和 P 两个矩阵的乘积去估计实际的评分矩阵，而且我们希望估计的评分矩阵 R 实际的评分矩阵不要相差太多，也就是求解下面的目标函数：

$$\hat{r}_{ui} = \overrightarrow{p_{uk}} \cdot \overrightarrow{q_{ik}} = \sum_{k=1}^{k} p_{uk} q_{ik}$$

这里涉及到最优化算法，在实际应用中，本文采用梯度下降法就可以求得这 P,Q 两个矩阵的估计值。对于评分预测我们利用平方差来构建损失函数：

$$Cost = \sum_{u,i \in R} \left(r_{ui} - \hat{r}_{ui}\right)^2 = \sum_{u,i \in R} \left(r_{ui} - \sum_{k=1}^{k} p_{uk} q_{ik}\right)^2$$

加入 L2 正则化：

$$Cost = \sum_{u,i \in R} \left(r_{ui} - \sum_{k=1}^{k} p_{uk} q_{ik}\right)^2 + \aleph\left(\sum_{U} p_{uk}^2 + \sum_{I} p_{ik}^2\right)$$

梯度下降更新参数，计算出最终的用户向量和物品向量。

3. 增量矩阵分解

在模型训练完以后，就会得到训练集里每个用户的向量和每个物品的向量。在实际的场景中，往往会不断有新的用户出现[17]，在训练集里没出现过，但是我们有他的历史行为数据，那我们如何计算该用户的预测评分呢？当然，最简单的方法就是把这个用户的行为数据合并到旧的训练集里，重新做一次矩阵分解，进而得到这个用户的向量[18]，但是这样做计算代价太大了，在时间上不可行。

为了解决训练数据集以外的用户(我们称之为新用户)的推荐问题，我们就需要用到增量矩阵分解算法。增量矩阵分解算法能根据用户历史行为数据，在不重算所有用户向量的前提下，快速计算出新用户向量[19]。

在梯度下降算法里，当固定 Q 计算 Pu 时，我们只需要用到用户 u 的历史行为数据 Rui，不同用户之间 Xu 的计算是相互独立的。这就启发我们，对于训练集以外的用户，我们同样可以用他的历史行为数据以及训练集上收敛时学到的 Q，来计算新用户的用户向量。

设用户历史行为数据为 Rui，训练集上学到的物品矩阵为 Q，要求解的用户向量为 Pu，我们可以将用户的历史行为数据 Rui 以及训练集上学习到的物品矩阵 Q，在保证 Q 不变的情况下，通过梯度下降算法，不断收敛得到用户 u 的用户向量 Pu，再通过算法 $Rui = Pu * Q$ 计算出用户对于所有物品的预测评分。

事实上，增量矩阵分解的目标函数中的 Q 也不一定要是矩阵分解在训练集上学出来的，只要 Q 中的每个向量都能表示对应物品的特征就行，也就是说可以是由其他数据和其他算法事先学出来的。

4. 实验分析

4.1. 数据集

本文并没有采用传统的推荐系统评估方式，没有将数据集分为测试数据集以及训练数据集，用训练

数据集进行训练，测试数据集进行系统测试。本文主要解决的问题是数据集之外的用户评分预测问题，传统的系统评估方式无法适用。

本文采用的数据样本为 mark 中用户的行为数据，分别测试将新用户的数据合并到旧的数据集中重新进行矩阵分解以及增量矩阵分解两种方式。数据细节见表 3。

Table 3. Mark data
表 3. Mark 数据

数据集	数量
users	2176
movies	9155
ratings	524,667

4.2. 实验设置

实验环境见表 4。

Table 4. Experimental environment
表 4. 实验环境

实验环境	配置信息
操作系统	macOS 10.15.6
内存	8GB
编程语言	Python 3.7.4

4.3. 实验结果

本文使用模型更新时间(Update Time)和召回率(Recall)作为实验的评价指标。其中 Recall@N 表示推荐 N 个物品时的召回率。实验结果见表 5。

Table 5. Experimental result
表 5. 实验结果

数据集	模型	Recall	Update Time (ms)
Mark Movies 500 K	矩阵分解	0.142	111,539
	增量矩阵分解	0.133	119
Movie Lens 1 M	矩阵分解	0.132	310,680
	增量矩阵分解	0.126	133
Movie Tweetings 100 K	矩阵分解	0.178	21680
	增量矩阵分解	0.142	43

模型	Recall	Update Time(ms)
矩阵分解	0.142	111,539
增量矩阵分解	0.133	119

从实验结果可以看出，本文提出的增量矩阵分解数据更新时间远远低于将数据合并到旧的数据集重新进行矩阵分解这种方式，在实际项目中，也能有很好的用户体验。可以高效的处理当一个新用户来到系统中的个性化推荐问题。

5. 结束语

现推荐系统模型大部分都是采用离线训练的方式[20]，模型训练完成以后再上线使用，在系统积累了一定的数据之后，或者固定一段时间后再将新的数据合并到旧的数据集中重新进行模型训练[21]。这样子的方式对于数据的实时性，用户的体验来说是有不太友好的[22]。本文提出的增量矩阵分解算法在解决数据集之外的新用户的评分预测问题有着很显著的效果，可以有效的解决新用户进入系统后数据快速更新的问题。每当有新的数据产生，新的用户产品的时候，系统可以快速的对其进行训练，使得模型能够实时应用，优化用户体验。

参考文献

[1] Anyosa, S.C., Vinagre, J. and Jorge, A.M. (2018) Incremental Matrix Co-Factorization for Recommender Systems with Implicit Feedback. *Companion Proceedings of the Web Conference*, Lyon, 23-27 April 2018, 1413-1418. https://doi.org/10.1145/3184558.3191585

[2] Ke, G., Meng, Q., Finley, T., *et al*. (2017) LightGBM: A Highly Efficient Gradient Boosting Decision Tree. *NIPS* 2017, Long Beach, 4-9 December 2017, 3146-3154.

[3] Covington, P., Adams, J. and Sargin, E. (2016) Deep Neural Networks for YouTube Recommendations. *Proceedings of the* 10*th ACM Conference on Recommender Systems*, Boston, 15-19 September 2016, 191-198. https://doi.org/10.1145/2959100.2959190

[4] Pennington, J., Socher, R. and Manning, C. (2014) Glove: Global Vectors for Word Representation. *Proceedings of the* 2014 *Conference on Empirical Methods in Natural Language Processing* (*EMNLP*), Doha, October 2014, 1532-1543. https://doi.org/10.3115/v1/D14-1162

[5] Mairal, J., Bach, F., Ponce, J., *et al*. (2010) Online Learning for Matrix Factorization and Sparse Coding. *Journal of Machine Learning Research*, **11**, 19-60.

[6] Pilászy, I., Zibriczky, D. and Tikk, D. (2010) Fast Als-Based Matrix Factorization for Explicit and Implicit Feedback Datasets. *Proceedings of the Fourth ACM Conference on Recommender Systems*, Barcelona, 26-30 September 2010, 71-78. https://doi.org/10.1145/1864708.1864726

[7] Johnson, C.C. (2014) Logistic Matrix Factorization for Implicit Feedback Data. *NIPS* 2014, Montreal, 13 December 2014, 27.

[8] He, X., Zhang, H., Kan, M.Y., *et al*. (2016) Fast Matrix Factorization for Online Recommendation with Implicit Feedback. *SIGIR*'16, Pisa, 17-21 July 2016, 549-558. https://doi.org/10.1145/2911451.2911489

[9] Zhang, S., Yao, L., Sun, A., *et al*. (2019) Deep Learning Based Recommender System: A Survey and New Perspectives. *ACM Computing Surveys* (*CSUR*), **52**, 5. https://doi.org/10.1145/3285029

[10] Mikolov, T., Chen, K., Corrado, G.S. and Dean, J. (2013) Efficient Estimation of Word Representations in Vector Space. *Proceedings of Workshop at ICLR*, Scottsdale, 2-4 May 2013.

[11] Mikolov, T., Sutskever, I., Chen, K., *et al*. (2013) Distributed Representations of Words and Phrases and Their Compositionality. *NIPS* 2013, Lake Tahoe, 5-8 December 2013, 3111-3119.

[12] Bell, R. and Koren, Y. (2007) Scalable Collaborative Filtering with Jointly Derived Neighborhood Interpolation Weights. *Seventh IEEE International Conference on Data Mining* (*ICDM* 2007), Omaha, 28-31 October 2007,43-52. https://doi.org/10.1109/ICDM.2007.90

[13] Zhou, Y., *et al*. (2008) Large-Scale Parallel Collaborative Filtering for the Netflix Prize. In: Fleischer R. and Xu J., Eds, *Algorithmic Aspects in Information and Management*, Springer, Berlin, 337-348. https://doi.org/10.1007/978-3-540-68880-8_32

[14] Chen, T. and Guestrin, C. (2016) Xgboost: A Scalable Tree Boosting System. *SIGKDD* 2016, San Francisco, 13-17 August 2016, 785-794. https://doi.org/10.1145/2939672.2939785

[15] Hu, Y.F., Koren, Y. and Volinsky, C. (2008) Collaborative Filtering for Implicit Feedback Datasets. 2008 *Eighth IEEE International Conference on Data Mining*, Pisa, 15-19 December 2008, 263-272.

[16] Goldberg, D., *et al*. (1992) Using Collaborative Filtering to Weave an Information Tapestry. *Communications of the ACM*, **35**, 61-70. https://doi.org/10.1145/138859.138867

[17] Sarwar, B.M., *et al*. (2000) Application of Dimensionality Reduction in Recommender System—A Case Study. In: *Proceedings KDD Workshop on Web Mining for e-Commerce*: *Challenges and Opportunities* (*WebKDD*), ACM Press, New York. https://doi.org/10.21236/ADA439541

[18] Funk, S. (2006) Netflix Update: Try This at Home. http://sifter.org/~simon/journal/20061211.html

[19] Koren, Y. (2008) Factorization Meets the Neighborhood: A Multifaceted Collaborative Filtering Model. In: *Proceedings* 14*th ACM SIGKDD Knowledge Discovery and Data Mining*, ACM Press, New York, 426-434. https://doi.org/10.1145/1401890.1401944

[20] Paterek, A. (2007) Improving Regularized Singular Value Decomposition for Collaborative Filtering. In: *Proceedings KDD Cup and Workshop*, ACM Press, New York, 39-42.

[21] Takács, G., *et al*. (2007) Major Components of the Gravity Recommendation System. *SIGKDD Explorations*, **9**, 80-84. https://doi.org/10.1145/1345448.1345466

[22] Salakhutdinov, R. and Mnih, A. (2008) Probabilistic Matrix Factorization. In: *Proceedings Advances in Neural Information Processing Systems* 20 (*NIPS* 07), ACM Press, New York, 1257-1264.