算法研究报告

2022-11-27 09:37:00

# 推荐系统之矩阵分解算法

**基本要求**  
- [x] 掌握课程要求的矩阵、图的数据结构和算法 - [x] 理解并复现Matrix Factorization算法 - [x] 撰写算法研究报告  
**进阶要求** - [x] 实现TransE算法 - [ ] 比较Matrix Factorization和TransE算法 - [ ] 改进Matrix Factorization算法 \*\*\*\*

## Matrix Factorization算法

### Introduction

推荐系统的矩阵分解算法是一种常用的推荐算法，它可以用于推荐物品或内容给用户。矩阵分解算法的基本思想是，将用户对物品的偏好表示为一个稀疏矩阵，然后对这个矩阵进行分解，得到两个矩阵：一个表示用户特征的矩阵，另一个表示物品特征的矩阵。经过这样的分解，矩阵分解算法可以学习出每个用户和物品的隐含特征，并基于这些特征来预测用户对某个物品的偏好程度。

矩阵分解算法通常采用梯度下降法来对用户和物品的特征矩阵进行训练。在训练过程中，算法会不断调整这两个矩阵，使得预测出的用户对物品的偏好值尽可能接近实际值。矩阵分解算法通常采用均方误差（Mean Squared Error，MSE）来衡量预测值与实际值之间的误差。通过不断调整算法的参数，使得 MSE 最小，就可以得到训练得到的用户和物品的特征矩阵。

一旦训练完成，矩阵分解算法就可以用于预测用户对未知物品的偏好程度。具体来说，假设已经训练得到了一个用户特征矩阵 U 和一个物品特征矩阵 V。如果要预测用户 u 对物品 i 的偏好值，可以通过计算 和 的内积来得到预测值。公式表示为：

矩阵分解算法由于其简单、高效，在推荐系统领域中得到了广泛应用。它能够从海量的数据中学习出用户和物品的隐含特征，并基于这些特征来推荐物品给用户。目前，矩阵分解算法是推荐系统领域中最为流行的方法之一。

除了简单、高效之外，矩阵分解算法还具有其他一些优点。例如，矩阵分解算法可以处理稀疏数据，因为它只需要用户对一部分物品的偏好信息，就可以预测用户对其他物品的偏好。此外，矩阵分解算法可以用于推荐多种类型的物品，例如电影、音乐、书籍等。这些优点使得矩阵分解算法在推荐系统领域中得到了广泛应用。

尽管矩阵分解算法具有许多优点，但它也存在一些局限性。例如，矩阵分解算法假设用户和物品之间存在一种线性关系，但实际上并非总是如此。此外，矩阵分解算法对于那些没有很多互动数据的用户和物品，其预测效果可能会变差。因此，矩阵分解算法在某些情况下可能会出现偏差。

为了克服矩阵分解算法的局限性，研究人员也提出了一些改进的算法。例如，有些算法采用了非线性模型来提高预测精度，例如深度神经网络。此外，还有一些算法引入了用户和物品的额外信息，例如用户的社交关系、物品的标签等，以提高推荐的准确性。

总之，矩阵分解算法是一种重要的推荐算法，它能够从海量的数据中学习出用户和物品的隐含特征，并基于这些特征来推荐物品给用户。尽管它存在一些局限性，但目前为止，它仍然是一种非常有效的算法。

### 推荐系统算法

从推荐系统做推荐的依据，[Matrix Factorization Techniques for Recommender Systems](https://doi.org/10.1109/MC.2009.263) 将推荐系统分为两种： 1. 基于内容(Content Filtering) 2. 协同过滤(Collaborative Filtering)

#### 基于内容的推荐算法

根据个人身份信息或者回答相关问题，来构造用户的特征。对于物品，则根据物品自身的内容，或属性来构造特征。例如电影，其特征可以是类型、风格、参演演员等等。有了用户信息和物品信息之后，将两者特征向量化，然后用某种策略，来给各个用户匹配合适的物品。

基于内容的推荐系统，需要较多的领域知识。用户和物品的特征需要针对不同场景来选择和设计。

#### 协同过滤推荐读法

协同过滤算法依赖于用户过去的行为信息，过去的购买记录、点赞记录、评分等等。协同过滤类的算法往往和领域无关，因为它不直接分析用户和物品自身的属性，只是基于用户与物品之间的交互信息（用户行为）来生成推荐。

协同过滤可以分成两大类：

1. Neighborhood Methods 这类方法会寻找相似用户或相似物品，以相似关系为依据来生成推荐。包括 Item-based CF 和 User-based CF 两类。
2. Latent Factor Models latent factor models 也基于 user-item 评分矩阵，但它并不用此矩阵来计算 user 或 item 间的相似度。而是用这个矩阵找出隐因素（factors），比如在电影推荐领域，喜剧、悲剧、动作、情感等都是会影响用户是否喜欢某特征。

* latent factor models 通常采用矩阵分解的方法，将 user-item 评分矩阵分解为 user 和 item 矩阵。
* 矩阵分解示意图|来源[https://cloud.tencent.com/developer/article/1442905]
* user 矩阵的各行是 user 的向量表示 ，item 矩阵各列是 item 的向量表示 。user 和 item 向量的每一维代表一种隐因子的强度。矩阵分解时可以控制 user 和 item 向量的维度，即控制隐因子的数量。
* 假如第一维代表的隐因子是电影的喜剧，那么 user 向量第一维的值表示该用户喜欢喜剧的程度，而 item 向量的第一维代表电影的喜剧成分的多少。
* user 和 item 向量的內积就是 user 对 item 的评分，可以看出 user 和 item 向量越契合，即 item 的各种特征恰好是 user 喜欢的，那么评分就高。

### 矩阵分解算法

矩阵分解的策略有很多，常见的有 SVD (Singular Value Decomposition)，NMF (Nonnegative Matrix Factorization) 等。

然而，很多情况下 user-item 矩阵都是稀疏矩阵（获得的数据不完整），所以直接采用线性代数中的矩阵分解算法是不可行的。一种想法就是直接利用矩阵里的已有值（训练集，也就是目前收集到的数据）来训练更新期望值 使其尽可能接近数据。得到训练好后的user矩阵和item矩阵后就可以得到完整的user-item矩阵，以此预测user先前没有评分过的item情况。

这样下来，我们的任务转变成下列优化问题：

其中 是已知的(user, item)集合（也就是训练集），后半部分是对学习到的参数进行正则化来避免过拟合的一个措施，常数 控制正则化的程度并且通过交叉验证确定。

对于上面的优化问题，可以有以下两种策略：

1. 随机梯度下降法(Stochastic gradient descent, SGD) 为方便起见定义： ； 为优化函数的负梯度方向（也就是下降最快方向），得到参数更新公式：
2. 交替最小二乘法(Alternating least squares, ALS) 因为  和  都是未知的，前面的优化目标，公式 (2) 是非凸函数，不好求解。但是如果能够固定  和  中的一个，交替地更新另外一个，公式中只有一个变量，而且是二次的，优化问题就更容易得到最优解。SGD 更容易实现且更快，但 ALS 可以并行化独立更新  和 。

### 矩阵分解算法的优化策略

#### ADDING BIASES

我们把评分值分解为4部分：global average, item bias, user bias 和 user-item interaction。举个例子，已知所有电影的平均评分是 3.7 分，而 *Titanic* 是不错的电影，会比平均分高，其 item bias 为 +0.5，另外 Joe 是一个严格的人，一向打分就偏低，存在 user bias -0.3。因此 Joe 对 *Titanic* 的评分为：

这里用户的偏差(Biases)就是考虑到不同用户对评分的严格程度不同，例如有的用户整体打分偏高，对烂片打分4.3分，对好片打分9.8分；有些电影因为某些流量明星的加入，使得对应的粉丝全体对该片打分偏高，而相应的黑粉就对此打分偏低……这些都是影响整体评分的偏差。

因此，我们对评分的函数更新为：

其中的四个参数对应我们对评分值的分解

优化目标转变成：

#### ADDITIONAL INPUT SOURCES 很多用户可能只对个别物品进行了评分，这就很难得出可靠的用户向量表示。引入其他的信息能够解决这种信息较少的问题。

推荐系统可以利用隐式信息，比如用户的浏览记录、搜索记录、鼠标停留信息等，在没有足够多的明确信息（购买、评分）时，此类信息也能在一定程度上对用户进行刻画。

考虑到上面这些，作者引入用户的 和 等信息。

1. Implicit feedback implicit feedback 指的是浏览记录、搜索记录等。定义 为用户有过 implicit feedback 的 items 集合，每一个 item 对应一个向量 中的 items 给用户 带来的特征可以表示为:

* 前面的 用于归一化。

1. User attributes 另外用户自身的属性也是一个信息来源，设用户有一组特征 ，每个特征用向量表示 ，用户的属性给用户来的特征可以表示为:

* 如此以来，用户对物品的评分可以表示为：
* 和 比起来，就相当于 做了些调整。

#### TEMPORAL DYNAMICS

有很多因素会随时间变化，比如用户看的电影越来越多，眼光越来越刁钻，以前喜欢给电影打 4 星，现在倾向于打 3 星。引入时序信号，可以捕获到用户或物品对应的矩阵对于时间的改变。引入时序信号后，对 的估计变为：

是物品的偏差，他会随着时间变化。比如速度与激情第一部刚刚上映时好评如潮，但是随着题材不断固化，以及后续加入的家庭因素冲淡了前几部打下的标签，人们对这系列电影评分逐渐变低。

是用户的偏差，可以理解为用户的品味会随着时间变化

是用户向量，表示用户对于物品的喜好会发生变化，系统会根据用户的隐性数据进行更新。比如之前喜欢看喜剧片，现在喜欢看战争片。

是隐因素向量，表示电影的不随时间变化的固有属性。

这里是使用过去的部分信息，来预测过去的另一部分信息。但在实际的推荐系统中，需要用过去的数据预测未来的用户的评分。模型需要定期重新训练，以尽可能准确地预测用户在接下来的一端实际的评分。

#### INPUT WITH VARYING CONFIDENCE LEVELS

不是所有评分都有一样的权重，有些评分可能受到了广告的影响，这对刻画长期的特征贡献不大。因此，作者对每个观察到的评分引入了 confidence level，然置信度低的评分贡献小一点。如此，优化目标变为：

### Matrix Factorization算法的简单实现

我们根据论文和github仓库用python简单实现了该算法

# 定义用户和物品的嵌入向量维度  
user\_dim = 100  
item\_dim = 100  
  
# 定义用户和物品的嵌入向量  
user\_embedding = np.random.random((num\_users, user\_dim))  
item\_embedding = np.random.random((num\_items, item\_dim))  
  
# 定义Matrix Factorization算法的损失函数  
def matrix\_factorization\_loss(user, item):  
 user\_embedding = user\_embedding[user]  
 item\_embedding = item\_embedding[item]  
 return np.sum((user\_embedding.dot(item\_embedding) - ratings[user][item]) \*\* 2)  
  
# 使用梯度下降法优化损失函数  
for i in range(num\_iterations):   
 user, item = generate\_rating()   
 loss = matrix\_factorization\_loss(user, item)   
 grad\_user, grad\_item = compute\_gradients(user, item)   
 user\_embedding[user] -= learning\_rate \* grad\_user   
 item\_embedding[item] -= learning\_rate \* grad\_item

上面的代码演示了Matrix Factorization算法的基本流程，包括定义嵌入向量、定义损失函数以及使用梯度下降法优化损失函数。在这个实例中，Matrix Factorization算法的损失函数是计算用户对物品的预测评分与实际评分的平方差，然后使用梯度下降法来优化损失函数，从而让用户和物品的嵌入向量更加满足约束条件。

实际应用中，Matrix Factorization算法的损失函数和优化方法可能会有所不同，但基本流程是相似的。Matrix Factorization算法的优点在于它可以有效地预测用户对物品的喜爱程度，并且模型参数数量随着用户和物品数量的增加而线性增长。但它也有一些缺点，比如对于稀疏数据可能表现不够理想。

## TransE算法

### Introduction

TransE 算法是一种将实体和关系表示为低维向量的方法，它可以用于基于三元组的知识图谱构建。TransE 的基本假设是，对于一个给定的实体和关系，它们的向量表示应该能够通过将实体向量相加得到关系向量。例如，假设有一个三元组，其中 和 是实体， 是实体 和 之间的关系。TransE 算法假设，实体 和 的向量表示分别为 和 ，关系 的向量表示为 。那么，TransE 算法假设在这种情况下，实体 和 的关系应该满足 。

TransE 算法可以通过将实体和关系向量投影到同一个低维空间中来实现这一假设。在模型训练过程中，TransE 算法会根据已知的三元组对实体和关系向量进行调整，以便满足 的假设。TransE 算法通过最小化训练样本中三元组的错误预测概率来实现这一目标。

TransE 算法在知识图谱构建领域中具有重要作用，它可以用于从给定的三元组中学习实体和关系的向量表示。一旦模型训练完成，TransE 算法可以用于预测新的三元组是否合法，以及对已知三元组进行推理。例如，假设在知识图谱中已经存在三元组，其中 是“北京”，y 是“上海”， 是“在中国”，并且已经通过 TransE 算法训练得到了相应的向量表示。那么，TransE 算法可以用来预测新的三元组 是否合法，其中 是另一个实体与 和 之间的关系。如果 是“在亚洲”，那么 TransE 算法可以通过检查 的情况来判断这个三元组是否合法。

TransE 算法被广泛应用于知识图谱构建领域，它具有计算复杂度低、训练速度快、模型精度高等优点。TransE 算法与其他常用的知识图谱构建方法，如 TransR 和 TransH 算法，都有不同的假设和特点。目前，TransE 算法是知识图谱构建领域中最为流行的方法之一。

TransE 算法虽然在知识图谱构建领域中取得了巨大成功，但它也存在一些局限性。例如，TransE 算法只能处理一些简单的关系，而不能处理一些更复杂的关系。此外，TransE 算法假设实体和关系之间的语义关系可以通过向量线性变换来表示，但实际上并非总是如此。因此，TransE 算法在处理一些特定的问题时可能会出现偏差。

鉴于 TransE 算法的局限性，研究人员也提出了一些改进的算法，如 TransR 和 TransH 算法。这些改进的算法在保留 TransE 算法的优点的同时，也弥补了 TransE 算法的不足。例如，TransR 算法引入了一个新的矩阵来更好地处理实体和关系之间的非线性关系。TransH 算法通过将实体向量投影到一个超平面上，来处理实体和关系之间的高维语义关系。

总之，TransE 算法是一种重要的知识图谱构建方法，它在研究人员和工业界中都得到了广泛应用。尽管它存在一些局限性，但目前为止，它仍然是一种非常有效的算法。

### 知识图谱

一条知识图谱可以表示为一个三元组(head, label, tail)。举个例子：电子科技大学的校长是曾勇，表示成三元组是（电子科技大学，校长，曾勇）。前者是主体，中间是关系，后者是客体。主体和客体统称为实体（entity）。关系有一个属性，不可逆，也就是说主体和客体不能颠倒过来。

知识图谱链接起来成为一个图（graph），每个节点是一个一个实体，每条边是一个关系，或者说是一个事实（fact）。也就是有向图，主体指向客体。

为了发挥知识库的图（graph）性，也为了得到统计学习的优势，我们需要将知识库嵌入（embedding）到一个低维空间里（比如10、20、50维）。

Graph Embedding 的中心思想是找到一种映射函数，将图中的每个节点转换为低维稠密的嵌入表示，要求在图中相似的节点在低维空间距离相近。得到的表示向量可用于下游任务，如节点分类、链接预测、可视化等。

### TransE

#### 算法思想

TranE是一篇[Bordes等人2013年发表在NIPS上的文章](https://scholar.google.com/scholar?cluster=8246464320979790535&hl=en&as_sdt=0,5)提出的算法。它的提出，是为了解决多关系数据（multi-relational data）的处理问题。transE是基于平移假设的模型，规定当三元组 为真，那么有 成立。只需要在训练过程中不断调整向量 ，使得最终结果尽可能靠近 。

#### 模型

给定一个三元组集合 ,每个三元组表示为 ，其中 属于实体集 ，对应的embedding 表示为 ，且满足向量加法： 。

定义距离公式（可以取 或者 范数）：

定义损失函数：

其中， 表示大于 0 取原值，小于 0 取 0 ；γ 为一个大于 0 的超参数。

上式表示被破坏的三元组(corrupted triplets set)，其中 head 实体或者 tail 实体被随机实体替换作为对照组。 训练模型时，期望原三元组损失函数更小，被破坏的三元组损失函数更大。

在训练模型过程中用梯度下降法（SGD）最小化目标函数，从而得到最优化模型, 这里的优化是指使表示学习结果对知识的真实与否有较好的辨别能力。

用于最大间隔法。作用相当于是一个正确triple与错误triple之前的间隔修正，margin越大，则两个triple之前被修正的间隔就越大，则对于词向量的修正就越严格。

#### 训练过程

论文中的训练过程

TransE算法的训练过程大概可以分为三部分：

1. 根据输入的维度 随机初始化实体矩阵和关系矩阵并归一化。
2. 根据 对输入的三元组进行随机抽取正例，并根据这些正例进行负采样。
3. 根据目标函数计算梯度，更新实体矩阵和关系矩阵中的值。
4. 对每个实体进行归一化，重复步骤2，3，4直到模型达到预设训练终止条件

entity需要在每次更新前进行归一化，这是通过人为增加embedding的norm来防止Loss在训练过程中极小化。

#### TransE算法简单实例

我们根据论文中的计算过程使用python复现了一下这个算法：

# 定义实体和关系的嵌入向量维度  
entity\_dim = 100  
relation\_dim = 100  
  
# 定义实体和关系的嵌入向量  
entity\_embedding = np.random.random((num\_entities, entity\_dim))  
relation\_embedding = np.random.random((num\_relations, relation\_dim))  
  
# 定义TransE算法的损失函数  
def transE\_loss(head, relation, tail):  
 head\_embedding = entity\_embedding[head]  
 relation\_embedding = relation\_embedding[relation]  
 tail\_embedding = entity\_embedding[tail]  
 return np.sum(np.abs(head\_embedding + relation\_embedding - tail\_embedding))  
  
# 使用梯度下降法优化损失函数  
for i in range(num\_iterations):  
 head, relation, tail = generate\_triple()  
 loss = transE\_loss(head, relation, tail)  
 grad\_head, grad\_relation, grad\_tail = compute\_gradients(head, relation, tail)  
 entity\_embedding[head] -= learning\_rate \* grad\_head  
 relation\_embedding[relation] -= learning\_rate \* grad\_relation  
 entity\_embedding[tail] -= learning\_rate \* grad\_tail

上面的代码演示了TransE算法的基本流程，包括定义嵌入向量、定义损失函数以及使用梯度下降法优化损失函数。在这个实例中，TransE算法的损失函数是计算实体和关系的嵌入向量之间的欧几里得距离，然后使用梯度下降法来优化损失函数，从而让实体和关系的嵌入向量更加满足约束条件。

实际应用中，TransE算法的损失函数和优化方法可能会有所不同，但基本流程是相似的。TransE算法的优点在于它简单易实现，可以有效地将实体和关系嵌入到低维向量空间中。但它也有一些缺点，比如对于一些复杂的关系可能表现不够理想。