Thinking\_HW\_Sat\_Week4

Xiang Guo

**Thinking 1 奇异值分解SVD的原理是怎样的，都有哪些应用场景**

=====

SVD是矩阵分解的一种方法。矩阵分解，简单来说，就是把原来的大矩阵，近似分解成两个小矩阵的乘积，在实际推荐计算时不再使用大矩阵，而是使用分解得到的两个小矩阵。以下是SVD和矩阵分解的原理。

1 实对称方阵的矩阵分解

对于一个n×n实对称方阵A，如果存在一个向量v是矩阵A的特征向量，可以表示成下面的形式：

Av=λv

其中，λ是特征向量v对应的特征值。如果矩阵A有n个线性无关的特征向量，那么矩阵A可以分解为:

A=QΣQ−1

其中，Q是矩阵A的特征向量组成的n×n的方阵，Σ是对角矩阵，每一个对角线元素就是一个特征值。注意到，特征值分解是有局限的，这里的矩阵是方阵，实际应用中，常见矩阵并不全是方阵。下面介绍SVD，SVD可以对任意矩阵进行分解。

2 奇异值分解(SVD)

假设矩阵A是m×n的矩阵，则AA^T是m×m的方阵，A^TA是n×n的方阵，对这两个方阵矩阵分解：

AA^T=UΛ1U^T

A^TA=VΛ2V^T

其中，Λ1和Λ2是对角矩阵，且对角线上非零元素均相同，也就是说两个方阵有相同的非零特征值，令非零特征值为σ1,σ2,⋯,σk，注意，k≤m，k≤n。根据σ1,σ2,⋯,σk可以得到矩阵A的特征值为：λ1=σ1−−√,λ2=σ2−−√,⋯,λk=σk−−√

下面就可以得到奇异值分解的表达式为：

A=UΛV^T

其中，U是一个m，×m的矩阵，Σ是m×n的矩阵，除了主对角线上的元素外其余全为0，主对角线上的每个元素称为奇异值，V是一个n×n的矩阵。U和V都是酉矩阵，满足U^TU=I，V^TV=I。

**总而言之，SVD分解的形式为3个矩阵相乘，左右两个矩阵分别表示用户/项目隐含因子矩阵，中间矩阵为奇异值矩阵并且是对角矩阵，每个元素满足非负性，并且逐渐减小。因此我们可以只需要前 k 个因子来表示它而达到降维的效果。**

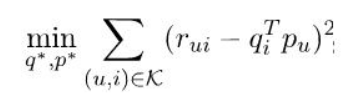
SVD一个常见的应用场景就是对特征降维，因为可以把一个特征矩阵降维，因为用SVD可以很容易得到任意矩阵的满秩分解，用满秩分解可以对数据做压缩。

**Thinking 2 funkSVD, BiasSVD，SVD++算法之间的区别是怎样的**

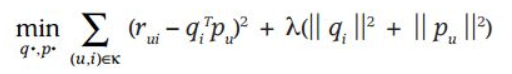
=====

**funkSVD**

普通的SVD算法首先需要填充矩阵，然后再进行分解降维，同时由于需要求逆操作（复杂度O(n^3)），存在计算复杂度高的问题，所以后来Simon Funk提出了FunkSVD的方法，它不在将矩阵分解为3个矩阵，而是分解为2个低秩的用户项目矩阵，同时降低了计算复杂度：



它借鉴线性回归的思想，通过最小化观察数据的平方来寻求最优的用户和项目的隐含向量表示。同时为了避免过度拟合（Overfitting）观测数据，又提出了带有L2正则项的FunkSVD：



**BiasSVD**

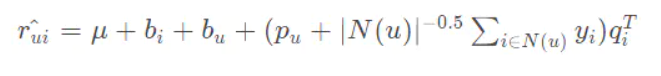
在FunkSVD提出来之后，陆续又提出了许多变形版本，其中相对流行的方法是BiasSVD，它是基于这样的假设：某些用户会自带一些特质，比如天生愿意给别人好评，心慈手软，比较好说话，有的人就比较苛刻，总是评分不超过3分（5分满分）；同时也有一些这样的项目，一被生产便决定了它的地位，有的比较受人们欢迎，有的则被人嫌弃，这也正是提出用户和项目偏置项的原因；项亮给出的解释是：对于一个评分系统有些固有属性和用户物品无关，而用户也有些属性和物品无关，物品也有些属性与用户无关，具体的预测公式如下：



其中，u为整个网站的平均评分，是真个网站的基调； b\_{u} 为用户的评分偏置，代表某个用户的评分基调， b\_{i} 为项目的被评分偏置，代表某个项目的属性基调。

SVD++

在用户除了显式评分外，隐式反馈信息同样有助于用户的偏好建模，因此随后提出了SVD++。它是基于这样的假设：用户除了对于项目的显式历史评分记录外，浏览记录或者收藏列表等**隐式反馈信息**同样可以从侧面一定程度上反映用户的偏好，比如用户对某个项目进行了收藏，可以从侧面反映他对于这个项目感兴趣，具体反映到预测公式为：



其中，|N(u)|表示用户u的行为物品集，yj表示物品j所表达的隐式反馈。

**Thinking 3 矩阵分解算法在推荐系统中有哪些应用场景，存在哪些不足**

=====

矩阵分解算法对于推荐系统来说存在两大场景即**评分预测（rating prediction）与Top-N推荐（item recommendation，item ranking**）。评分预测场景主要用于评价网站，比如用户给自己看过的电影评多少分（MovieLens），或者用户给自己看过的书籍评价多少分（Douban）。这也是矩阵分解算法最主要的应用场景。

Top-N推荐场景主要用于购物网站或者一般拿不到显式评分信息的网站，即通过用户的隐式反馈信息来给用户推荐一个可能感兴趣的列表以供其参考。可以使用SVD++的算法根据用户隐式反馈信息来进行评价打分然后再来做排序任务，**不过根据我自己调查下来的结果，一般这一类的应用场景可以直接使用排序模型来对其建模。因此，评分预测是矩阵分解的主要战场。**

不足之处就在于基础矩阵分解只用了userId和itemId两个维度的信息，所有学到的知识都蕴含在user向量和item向量中。可解释性相对来说比较一般，另一个不足的方面在于学习过程中很难融合更多有用的特征，比如用户的人口统计学信息，长期的类目、品牌和商家的偏好；以及商品的基础特征信息，如类目、品牌等。因而，矩阵分解的泛化能力受到一定的限制。

另外我还查到矩阵分解的可并行度比较低，不太确定这是什么意思。

**Thinking 4 假设一个小说网站，有N部小说，每部小说都有摘要描述。如何针对该网站制定基于内容的推荐系统，即用户看了某部小说后，推荐其他相关的小说。原理和步骤是怎样的**

=====

对于这类的基于内容的推荐系统，一般可以采用以下步骤来给用户推荐相关的小说。

1. 首先，提取用户看的小说的重要特征，然后计算这些特征的特征向量。
2. 对整个小说网站的所有N部小说提取特征矩阵。
3. TF-IDF可以帮我们抽取文本的重要特征，做成item embedding。
4. 最后，计算当前小说特征向量与整个小说网站小说特征矩阵的余弦相似度，取相似度最大的top k个来推荐给用户。

**Thinking5 Word2Vec的应用场景有哪些**

====

**在社交网络中的推荐**

给当前用户推荐他可能关注的『大V』。对一个新用户，此题基本无解，如果在已知用户关注了几个『大V』之后，相当于知道了当前用户的一些关注偏好，根据此偏好给他推荐和他关注过大V相似的大V，就是一个很不错的推荐策略。所以，如果可以求出来任何两个V用户的相似度，上面问题就可以基本得到解决。

我们知道word2vec中两个词的相似度可以直接通过余弦来衡量，接下来就是如何将每个V用户变为一个词向量的问题了。巧妙的地方就是如何定义doc和word，针对上面问题，可以将每一个大V 定义为一个词；将每个用户关注的大V，按照关注的顺序排列就是doc，形成文章。

**计算商品的相似度**

在商品推荐的场景中，竞品推荐和搭配推荐的时候都有可能需要计算任何两个商品的相似度，根据浏览/收藏/下单/App下载等行为，可以将商品看做词，将每一个用户的一类行为序看做一个文档，通过word2vec将其训练为一个向量。

同样的，在计算广告中，根据用户的点击广告的点击序列，将每一个广告变为一个向量。变为向量后，用此向量可以生成特征融入到rank模型中。

**作为另一个模型的输入（nlp应用）**

在nlp的任务中，可以通过将词聚类后，生成一维新的特征来使用。在CRF实体识别的任务中，聚类结果类似词性，可以作为特征来使用。

在依存句法分析的任务中，哈工大ltp的nndepparser则是将词向量直接作为输入。