

50 | 推荐系统（下）：如何通过SVD分析用户和物品的矩阵？

黄申 2019-04-10



00:00

讲述：黄申

大小：8.45M

09:13

你好，我是黄申。

上一节，我们讲了如何使用矩阵操作，实现基于用户或者物品的协同过滤。实际上，推荐系统是个很大的课题，你可以尝试不同的想法。比如，对于用户给电影评分的案例，是不是可以使用 SVD 奇异值的分解，来分解用户评分的矩阵，并找到“潜在”的电影主题呢？如果在一定程度上实现这个目标，那么我们可以通过用户和主题，以及电影和主题之间的关系来进行推荐。今天，我们继续使用 MovieLens 中的一个数据集，尝试 Python 代码中的 SVD 分解，并分析一些结果所代表的含义。

SVD 回顾以及在推荐中的应用

在实现 SVD 分解之前，我们先来回顾一下 SVD 的主要概念和步骤。如果矩阵 X 是对称的方阵，那么我们可以求得这个矩阵的特征值和特征向量，并把矩阵 X 分解为特征值和特征向量的乘积。

假设我们求出了矩阵 X 的 n 个特征值 $\lambda_1, \lambda_2, \dots, \lambda_n$ ，以及这 n 个特征值所对应的特征向量 v_1, v_2, \dots, v_n ，那么矩阵 X 可以表示为：

$$X = V \Sigma V^{-1}$$

其中， V 是这 n 个特征向量所张成的 $n \times n$ 维矩阵，而 Σ 是这 n 个特征值为主对角线的 $n \times n$ 维矩阵。这个过程就是特征分解（Eigendecomposition）。

如果我们会把 V 的这 n 个特征向量进行标准化处理，那么对于每个特征向量 V_i ，就有 $\|V_i\|_2 = 1$ ，而这表示 $V_i' V_i = 1$ ，此时 V 的 n 个特征向量为标准正交基，满足 $V' V = I$ ，也就是说， V 为酉矩阵，有 $V' = V^{-1}$ 。这样一来，我们就可以把特征分解表达式写作：

$$X = V \Sigma V'$$

可是，如果矩阵 X 不是对称的方阵，那么我们不一定能得到有实数解的特征分解。但是，SVD 分解可以避免这个问题。

我们可以把 X 的转置 X' 和 X 做矩阵乘法，得到一个 $n \times n$ 维的对称方阵 $X' X$ ，并对这个对称方阵进行特征分解。分解的时候，我们得到了矩阵 $X' X$ 的 n 个特征值和对应的 n 个特征向量 v ，其中所有的特征向量叫作 X 的右奇异向量。通过所有右奇异向量我们可以构造一个 $n \times n$ 维的矩阵 V 。

类似地，如果我们把 X 和 X' 做矩阵乘法，那么会得到一个 $m \times m$ 维的对称方阵 $X X'$ 。由于 $X X'$ 也是方阵，因此我们同样可以对它进行特征分解，并得到矩阵 $X X'$ 的 m 个特征值和对应的 m 个特征向量 u ，其中所有的特征向量叫作 X 的左奇异向量。通过所有左奇异向量我们可以构造一个 $m \times m$ 的矩阵 U 。

现在，包含左右奇异向量的 U 和 V 都求解出来了，只剩下奇异值矩阵 Σ 了。 Σ 除了对角线上是奇异值之外，其他位置的元素都是 0，所以我们只要求出每个奇异值 σ 就可以了。之前我们已经推导过， σ 可以通过两种方式获得。第一种方式是计算下面这个式子：

$$\sigma_i = \frac{X_{v_i}}{u_i}$$

其中 v_i 和 u_i 都是列向量。一旦我们求出了每个奇异值 σ ，那么就能得到奇异值矩阵 Σ 。

第二种方式是通过 $X' X$ 矩阵或者 $X X'$ 矩阵的特征值之平方根，来求奇异值。计算出每个奇异值 σ ，那么就能得到奇异值矩阵 Σ 了。

通过上述几个步骤，我们就能把一个 $m \times n$ 维的实数矩阵，分解成 $X = U \Sigma V'$ 的形式。那么这种分解对于推荐系统来说，又有怎样的意义呢？

之前我讲过，在潜在语义分析 LSA 的应用场景下，分解之后所得到的奇异值 σ ，对应一个语义上的“概念”，而 σ 值的大小表示这个概念在整个文档集合中的重要程度。 U 中的左奇异向量表示了每个文档和这些语义“概念”的关系强弱， V 中的右奇异向量表示每个词条和这些语义“概念”的关系强弱。

最终，SVD 分解把原来的“词条 - 文档”关系，转换成了“词条 - 语义概念 - 文档”的关系。而在推荐系统的应用场景下，对用户评分矩阵的 SVD 分解，能够帮助我们找到电影中潜在的“主题”，比如科幻类、动作类、浪漫类、传记类等等。


分解之后所得到的奇异值 σ 对应了一个“主题”， σ 值的大小表示这个主题在整个电影集合中的重要程度。 U 中的左奇异向量表示了每位用户对这些“主题”的喜好程度， V 中的右奇异向量表示每部电影和这些“主题”的关系强弱。

最终，SVD 分解把原来的“用户 - 电影”关系，转换成了“用户 - 主题 - 电影”的关系。有了这种新的关系，即使我们没有人工标注的电影类型，同样可以使用更多基于电影主题的推荐方法，比如通过用户对电影主题的评分矩阵，进行基于用户或者电影的协同过滤。

接下来，我会使用同样一个 MovieLens 的数据集，一步步展示如何通过 Python 语言，对用户评分的矩阵进行 SVD 分解，并分析一些结果的示例。


Python 中的 SVD 实现和结果分析

和上节的代码类似，首先我们需要加载用户对电影的评分。不过，由于非并行 SVD 分解的时间复杂度是 3 次方数量级，而空间复杂度是 2 次方数量级，所以对硬件资源要求很高。这里为了节省测试的时间，我增加了一些语句，只取大约十分之一的数据。

 复制代码


```
1 import pandas as pd
2 from numpy import *
3
4
5 # 加载用户对电影的评分数据
6 df_ratings = pd.read_csv("/Users/shenhuang/Data/ml-latest-small/ratings.csv")
7
8
9 # 获取用户的数量和电影的数量，这里我们只取前 1/10 来减小数据规模
10 user_num = int(df_ratings["userId"].max() / 10)
11 movie_num = int(df_ratings["movieId"].max() / 10)
12
13
14 # 构造用户对电影的二元关系矩阵
15 user_rating = [[0.0] * movie_num for i in range(user_num)]
16
17
18 i = 0
19 for index, row in df_ratings.iterrows(): # 获取每行的 index、row
20
21
22     # 由于用户和电影的 ID 都是从 1 开始，为了和 Python 的索引一致，减去 1
23     userId = int(row["userId"]) - 1
24     movieId = int(row["movieId"]) - 1
25
26
27     # 我们只取前 1/10 来减小数据规模
28     if (userId >= user_num) or (movieId >= movie_num):
29         continue
30
31
32     # 设置用户对电影的评分
33     user_rating[userId][movieId] = row["rating"]
34
```

之后，二维数组转为矩阵，以及标准化矩阵的代码和之前是一致的。

 复制代码

```
1 # 把二维数组转化为矩阵
2 x = mat(user_rating)
3
4
5 # 标准化每位用户的评分数据
6 from sklearn.preprocessing import scale
7
8
9 # 对每一行的数据，进行标准化
10 x_s = scale(x, with_mean=True, with_std=True, axis=1)
11 print(" 标准化后的矩阵: ", x_s)
12
```

Python 的 numpy 库，已经实现了一种 SVD 分解，我们只调用一个函数就行了。

 复制代码

```
1 # 进行 SVD 分解
2 from numpy import linalg as LA
3
4
5 u,sigma,vt = LA.svd(x_s, full_matrices=False, compute_uv=True)
6 print("U 矩阵: ", u)
7 print("Sigma 奇异值: ", sigma)
8 print("V 矩阵: ", vt)
9
```

最后输出的 Sigma 奇异值大概是这样的：

 复制代码

```
1 Sigma 奇异值: [416.56942602 285.42546812 202.25724866 ... 79.26188177 76.3516740
2
```

最后几个奇异值不是 0，说明我们没有办法完全忽略它们，不过它们相比最大的几个奇异值还是很小的，我们可以去掉这些值来求得近似的解。

为了验证一下 SVD 的效果，我们还可以加载电影的元信息，包括电影的标题和类型等等。我在这里使用了一个基于哈希的 Python 字典结构来存储电影 ID 到标题和类型的映射。

 复制代码

```
1 # 加载电影元信息
2 df_movies = pd.read_csv("/Users/shenhuang/Data/ml-latest-small/movies.csv")
3 dict_movies = {}
4
5
6 for index, row in df_movies.iterrows(): # 获取每行的 index、row
7     dict_movies[row["movieId"]] = "{0},{1}".format(row["title"], row["genres"])
8 print(dict_movies)
9
```

我刚刚提到，分解之后所得到的奇异值 σ 对应了一个“主题”， σ 值的大小表示这个主题在整个电影集合中的重要程度，而 V 中的右奇异向量表示每部电影和这些“主题”的关系强弱。所以，我们可以对分解后的每个奇异值，通过 V 中的向量，找找看哪些电影和这个奇异值所对应的主题更相关，然后看看 SVD 分解所求得的主题是不是合理。比如，我们可以使用下面的代码，来查看和向量 Vt_1 ，相关的电影主要有哪些。

 复制代码

```
1 # 输出和某个奇异值高度相关的电影，这些电影代表了一个主题
2 print(max(vt[1,:]))
3 for i in range(movie_num):
4     if (vt[1][i] > 0.1):
5         print(i + 1, vt[1][i], dict_movies[i + 1])
6
```

需要注意的是，向量中的电影 ID 和原始的电影 ID 差 1，所以在读取 dict_movies 时需要使用 (i + 1)。这个向量中最大的分值大约是 0.173，所以我把阈值设置为 0.1，并输出了所有分值大于 0.1 的电影，电影列表如下：

 复制代码

```
1 0.17316444479201024
2 260 0.14287410901699643 Star Wars: Episode IV - A New Hope (1977),Action|Adventu
3 1196 0.1147295905497075 Star Wars: Episode V - The Empire Strikes Back (1980),Ac
4 1198 0.15453176747222075 Raiders of the Lost Ark (Indiana Jones and the Raiders
5 1210 0.10411193224648774 Star Wars: Episode VI - Return of the Jedi (1983),Actio
6 2571 0.17316444479201024 Matrix, The (1999),Action|Sci-Fi|Thriller
7 3578 0.1268370902126096 Gladiator (2000),Action|Adventure|Drama
8 4993 0.12445203514448012 Lord of the Rings: The Fellowship of the Ring, The (200
9 5952 0.12535012292041953 Lord of the Rings: The Two Towers, The (2002),Adventure
10 7153 0.10972312192709989 Lord of the Rings: The Return of the King, The (2003),A
11
```

从这个列表可以看出，这个主题是关于科幻或者奇幻类的动作冒险题材。

使用类似的代码和同样的阈值 0.1，我们来看看和向量 Vt_5 ，相关的电影主要有哪些。

 复制代码

```
1 # 输出和某个奇异值高度相关的电影，这些电影代表了一个主题
2 print(max(vt[5,:]))
3 for i in range(movie_num):
4     if (vt[5][i] > 0.1):
5         print(i + 1, vt[5][i], dict_movies[i + 1])
6
```

电影列表如下：

 复制代码

```
1 0.13594520920117012
2 21 0.13557812349701226 Get Shorty (1995),Comedy|Crime|Thriller
3 50 0.11870851441884082 Usual Suspects, The (1995),Crime|Mystery|Thriller
4 62 0.11407971751480048 Mr. Holland's Opus (1995),Drama
5 168 0.10295400456394468 First Knight (1995),Action|Drama|Romance
6 222 0.12587492482374366 Circle of Friends (1995),Drama|Romance
7 261 0.13594520920117012 Little Women (1994),Drama
8 339 0.10815473505804706 While You Were Sleeping (1995),Comedy|Romance
```

```
9 357 0.11108191756350501 Four Weddings and a Funeral (1994),Comedy|Romance
10 527 0.1305895737838763 Schindler's List (1993),Drama|War
11 595 0.11155774544755555 Beauty and the Beast (1991),Animation|Children|Fantasy|M
12
```

从这个列表可以看出，这个主题更多的是关于剧情类题材。就目前所看的两个向量来说，SVD 在一定程度上区分了不同的电影主题，你也可以使用类似的方式查看更多的向量，以及对应的电影名称和类型。

总结

在今天的内容中，我们回顾了 SVD 奇异值分解的核心思想，解释了如何通过 XX' 和 $X'X$ 这两个对称矩阵的特征分解，求得分解后的 U 矩阵、 V 矩阵和 Σ 矩阵。另外，我们也解释了在用户对电影评分的应用场景下，SVD 分解后的 U 矩阵、 V 矩阵和 Σ 矩阵各自代表的意义，其中 Σ 矩阵中的奇异值表示了 SVD 挖掘出来的电影主题， U 矩阵中的奇异向量表示用户对这些电影主题的评分，而 V 矩阵中的奇异向量表示了电影和这些主题的相关程度。

我们还通过 Python 代码，实践了这种思想在推荐算法中的运用。从结果的奇异值和奇异向量可以看出，SVD 分解找到了一些 MovieLens 数据集上的电影主题。这样我们就可以把用户针对电影的评分转化为用户针对主题的评分。由于主题通常远远小于电影，所以 SVD 的分解也帮助我们实现了降低特征维度的目的。

SVD 分解能够找到一些“潜在的”因素，例如语义上的概念、电影的主题等等。虽然这样操作可以降低特征维度，去掉一些噪音信息，但是由于 SVD 分解本身的计算量也很大，所以从单次的执行效率来看，SVD 往往无法起到优化的作用。在这种情况下，我们可以考虑把它和一些监督式的学习相结合，使用一次分解的结果构建分类器，提升日后的执行效率。

思考题

刚才 SVD 分解实验中得到的 U 矩阵，是用户对不同电影主题的评分矩阵。请你使用这个 U 矩阵，进行基于用户或者基于主题（物品）的协同过滤。

欢迎留言和我分享，也欢迎你在留言区写下今天的学习笔记。你可以点击“请朋友读”，把今天的内容分享给你的好友，和他一起精进。



程序员的数学基础课

在实战中重新理解数学

黄申

LinkedIn 资深数据科学家



新版升级：点击「 请朋友读」，10位好友免费读，邀请订阅更有**现金**奖励。

©



一手资源 同步更新 加微信 ixuexi66

由作者筛选后的优质留言将会公开显示，欢迎踊跃留言。

Ctrl + Enter 发表

0/2000字

提交留言

精选留言

由作者筛选后的优质留言将会公开显示，欢迎踊跃留言。

