# 协同过滤推荐系统的Python实现

现今，****推荐系统****最普遍的推荐系统的类型是****基于内容****和****协同过滤（CF）****。协同过滤基于用户对产品的态度产生推荐。与此相反，基于内容的推荐系统集中于物品的属性，并基于它们之间的相似性为你推荐。一般情况下，协同过滤（CF）是推荐引擎的主力。该算法具有能够自身进行特征学习的一个非常有趣的特性，这意味着它可以开始学习使用哪些特性。CF可以分为****基于内存的协同过滤****和****基于模型的协同过滤****。这里使用奇异值分解（SVD）实现基于模型的CF和通过计算余弦相似实现基于内存的CF。

这里将使用MovieLens数据集，它是在实现和测试推荐引擎时所使用的最常见的数据集之一。它包含来自于943个用户以及精选的1682部电影的100K个电影打分。可以[在这里](http://files.grouplens.org/datasets/movielens/ml-100k.zip)下载数据集。

|  |  |
| --- | --- |
| 1  2 | import numpy **as** np      import pandas **as** pd |

读入****u.data****文件，它包含完整的数据集。可以[在这里](http://files.grouplens.org/datasets/movielens/ml-100k-README.txt)阅读该数据集的简要说明。

|  |  |
| --- | --- |
| 1  2 | header = ['user\_id', 'item\_id', 'rating', 'timestamp']      df = pd.read\_csv('ml-100k/u.data', sep='\t', names=header) |

先看看数据集中的前两行。接下来，让我们计算唯一用户和电影的数量。

|  |  |
| --- | --- |
| 1  2  3 | n\_users = df.user\_id.unique().shape[0]      n\_items = df.item\_id.unique().shape[0]      Print('Number of users = ' + str(n\_users) + ' | Number of movies = ' + str(n\_items)) |

|  |  |
| --- | --- |
| 1 | Number of users = 943 | Number of movies = 1682 |

你可以使用[scikit-learn](http://scikit-learn.org/stable/)库将数据集分割成测试和训练。[Cross\_validation.train\_test\_split](http://scikit-learn.org/stable/modules/generated/sklearn.cross_validation.train_test_split.html)根据测试样本的比例（test\_size），这里涉是0.25，来将数据混洗并分割成两个数据集。

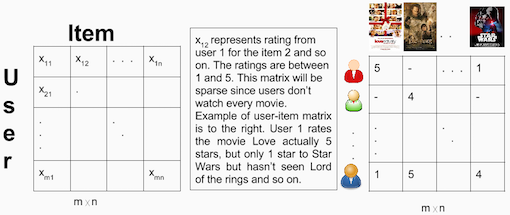
|  |  |
| --- | --- |
| 1  2 | from sklearn import cross\_validation **as** cv      train\_data, test\_data = cv.train\_test\_split(df, test\_size=0.25) |

# 1.基于内存的协同过滤

基于内存的协同过滤方法可以分为两个主要部分：****用户-产品协同过滤****和****产品-产品协同过滤****。一个用户-产品协同过滤将选取一个特定的用户，基于打分的相似性发现类似于该用户的用户，并推荐那些相似用户喜欢的产品。相比之下，产品-产品协同过滤会选取一个产品，发现喜欢该产品的用户，并找到这些用户或相似的用户还喜欢的其他的产品。输入一个产品，然后输出其他产品作为推荐。

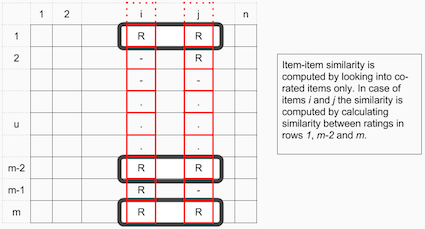
* 用户-产品协同过滤: “喜欢这个东西的人也喜欢……”
* 产品-产品协同过滤: “像你一样的人也喜欢……”

在这两种情况下，从整个数据集构建一个用户-产品矩阵。由于你已经将数据拆分到测试集和训练集，那么你将需要创建两个[943 x 1682]矩阵。训练矩阵包含75%的打分，而测试矩阵包含25%的打分。

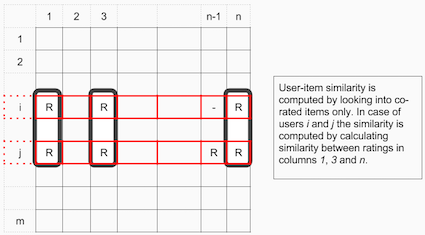
用户-产品矩阵的例子：[](https://camo.githubusercontent.com/717799387063c77489c7716314eaf621e3c66ed3/687474703a2f2f7333332e706f7374696d672e6f72672f61793074793930666a2f424c4f475f4343415f382e706e67)

在构建了用户-产品矩阵后，计算相似性并创建一个相似性矩阵。

在产品-产品协同过滤中的产品之间的相似性值是通过观察所有对两个产品之间的打分的用户来度量的。

[](https://camo.githubusercontent.com/8b1245121bdbe57c3aa67642cbcdff25a7fba08c/687474703a2f2f7333332e706f7374696d672e6f72672f693532326d6138337a2f424c4f475f4343415f31302e706e67)

对于用户-产品协同过滤，用户之间的相似性值是通过观察所有同时被两个用户打分的产品来度量的。

[](https://camo.githubusercontent.com/1f793e2f05b4abcfbfae708acd5bc0dcc26d27c0/687474703a2f2f7333332e706f7374696d672e6f72672f6d6c68337a337a34662f424c4f475f4343415f31312e706e67)

通常用于推荐系统中的距离矩阵是余弦相似性，其中，打分被看成n维空间中的向量，而相似性是基于这些向量之间的角度进行计算的。用户a和m的余弦相似性可以使用下面的公式进行计算，其中，获取用户向量的点积[IMG_259](https://camo.githubusercontent.com/b78ad5cf952ede7731d6c33bdb5f951817a9583a/68747470733a2f2f6c617465782e636f6465636f67732e636f6d2f6769662e6c617465783f24755f6b24)和[IMG_260](https://camo.githubusercontent.com/446e0d24860317ea3ff3f956b59f8d029ae391b9/68747470733a2f2f6c617465782e636f6465636f67732e636f6d2f6769662e6c617465783f24755f6124)，然后用向量的欧几里得长度的乘积来除以它。

[IMG_261](https://camo.githubusercontent.com/2708a4ee2dbe0c28e3290760d362c9587609b651/68747470733a2f2f6c617465782e636f6465636f67732e636f6d2f6769662e6c617465783f735f75253545253742636f7325374428755f6b2c755f61293d25354366726163253742755f6b2673706163653b25354363646f742673706163653b755f612673706163653b2537442537422673706163653b2535436c6566742673706163653b2535432537432673706163653b755f6b2673706163653b25354372696768742673706163653b2535432537432673706163653b2535436c6566742673706163653b2535432537432673706163653b755f612673706163653b25354372696768742673706163653b2535432537432673706163653b2537442673706163653b3d2535436672616325374225354373756d2673706163653b785f2537426b2c6d253744785f253742612c6d2537442537442537422535437371727425374225354373756d2673706163653b785f2537426b2c6d2537442535453225354373756d2673706163653b785f253742612c6d25374425354532253744253744)

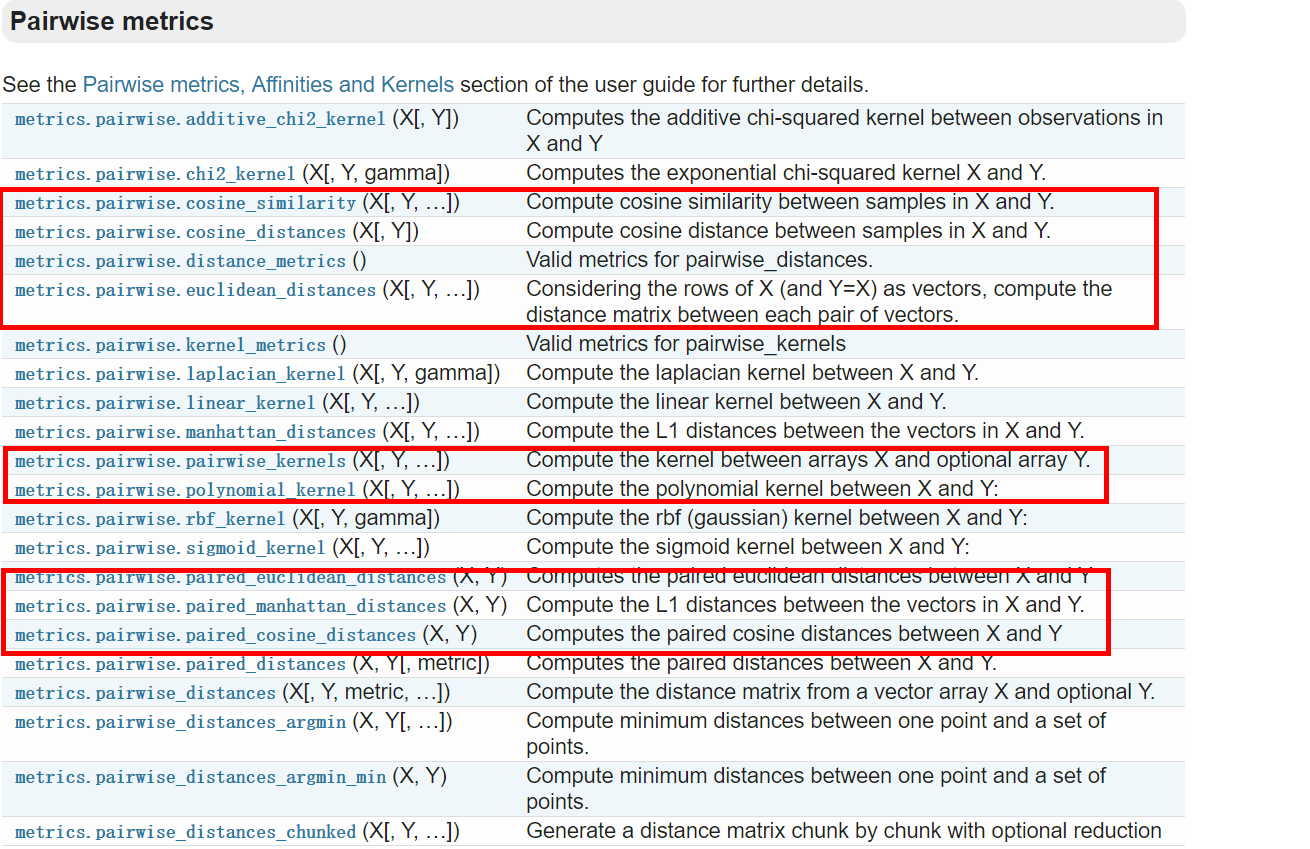
要计算产品m和b之间的相似性，使用公式：

[IMG_262](https://camo.githubusercontent.com/75cf08beab6b7af134d3af71c26afb18acf6fc15/68747470733a2f2f6c617465782e636f6465636f67732e636f6d2f6769662e6c617465783f735f75253545253742636f7325374428695f6d2c695f62293d25354366726163253742695f6d2673706163653b25354363646f742673706163653b695f622673706163653b2537442537422673706163653b2535436c6566742673706163653b2535432537432673706163653b695f6d2673706163653b25354372696768742673706163653b2535432537432673706163653b2535436c6566742673706163653b2535432537432673706163653b695f622673706163653b25354372696768742673706163653b2535432537432673706163653b2537442673706163653b3d2535436672616325374225354373756d2673706163653b785f253742612c6d253744785f253742612c622537442537442537422535437371727425374225354373756d2673706163653b785f253742612c6d2537442535453225354373756d2673706163653b785f253742612c6225374425354532253744253744)

第一步是创建用户-产品矩阵。由于你既有测试数据，又有训练数据，那么你需要创建两个矩阵。

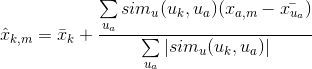
|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | #Create two user-item matrices, one for training and another for testing      train\_data\_matrix = np.zeros((n\_users, n\_items))  **for** line **in** train\_data.itertuples():          train\_data\_matrix[line[1]-1, line[2]-1] = line[3]        test\_data\_matrix = np.zeros((n\_users, n\_items))  **for** line **in** test\_data.itertuples():          test\_data\_matrix[line[1]-1, line[2]-1] = line[3] |

你可以使用sklearn的[pairwise\_distances](http://scikit-learn.org/stable/modules/generated/sklearn.metrics.pairwise.pairwise_distances.html)函数来计算余弦相似性。注意，输出范围从0到1，因为打分都是正的。



|  |  |
| --- | --- |
| 1  2  3 | from sklearn.metrics.pairwise import pairwise\_distances      user\_similarity = pairwise\_distances(train\_data\_matrix, metric='cosine')      item\_similarity = pairwise\_distances(train\_data\_matrix.T, metric='cosine') |

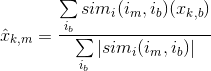
下一步是做出预测。你已经创建了相似性矩阵：user\_similarity和item\_similarity，因此，你可以通过为基于用户的CF应用下面的公式做出预测：

[](https://camo.githubusercontent.com/3e27f6a8a829d90129e301b287e18835c82a526c/68747470733a2f2f6c617465782e636f6465636f67732e636f6d2f6769662e6c617465783f253543686174253742782537445f2537426b2c6d2537442673706163653b3d2673706163653b253543626172253742782537445f2537426b2537442673706163653b26706c75733b2673706163653b2535436672616325374225354373756d2535436c696d6974735f253742755f612537442673706163653b73696d5f7528755f6b2c2673706163653b755f61292673706163653b28785f253742612c6d2537442673706163653b2d2673706163653b253543626172253742785f253742755f612537442537442925374425374225354373756d2535436c696d6974735f253742755f6125374425374373696d5f7528755f6b2c2673706163653b755f6129253743253744)

你可以将用户k和a之间的相似性看成权重，它乘以相似用户a (校正的平均评分用户)的评分。你需要规范化该值，使打分位于1到5之间，最后，对你尝试预测的用户的平均评分求和。

这里的想法是，某些用户可能会倾向于对所有的电影，总是给予高或低评分。这些用户提供的评分的相对差比绝对评分值更重要。举个例子：假设，用户k对他最喜欢的电影打4星，而对所有其他的好电影打3星。现在假设另一个用户t对他/她喜欢的电影打5星，而对他/她感到无聊的电影打3星。那么这两个用户可能品味非常相似，但对打分系统区别对待。

当为基于产品的CF进行预测时，你无须纠正用户的平均打分，因为查询用户本事就是用来做预测的。

[](https://camo.githubusercontent.com/a0e4650478d4da38c2961c48ce8aa4743cf7e499/68747470733a2f2f6c617465782e636f6465636f67732e636f6d2f6769662e6c617465783f253543686174253742782537445f2537426b2c6d2537442673706163653b3d2673706163653b2535436672616325374225354373756d2535436c696d6974735f253742695f622537442673706163653b73696d5f6928695f6d2c2673706163653b695f62292673706163653b28785f2537426b2c62253744292673706163653b25374425374225354373756d2535436c696d6974735f253742695f6225374425374373696d5f6928695f6d2c2673706163653b695f6229253743253744)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | def predict(ratings, similarity, type='user'):  **if** type == 'user':              mean\_user\_rating = ratings.mean(axis=1)              #You use np.newaxis so that mean\_user\_rating has same format as ratings              ratings\_diff = (ratings - mean\_user\_rating[:, np.newaxis])              pred = mean\_user\_rating[:, np.newaxis] + **similarity.dot(ratings\_diff) / np.array([np.abs(similarity).sum(axis=1)]).T**          elif type == 'item':              pred = **ratings.dot(similarity) / np.array([np.abs(similarity).sum(axis=1)])**  **return** pred |

|  |  |
| --- | --- |
| 1  2 | item\_prediction = predict(train\_data\_matrix, item\_similarity, type='item')      user\_prediction = predict(train\_data\_matrix, user\_similarity, type='user') |

## 1.1评估

有许多评价指标，但其中最受欢迎的用来度量预测评分的准确性的指标是均方根误差 (RMSE)。

[IMG_265](https://camo.githubusercontent.com/da51be69e6899ef7a52095d1e9f5b6479db51497/68747470733a2f2f6c617465782e636f6465636f67732e636f6d2f6769662e6c617465783f524d53452673706163653b3d2535437371727425374225354366726163253742312537442537424e2537442673706163653b25354373756d2673706163653b28785f692673706163653b2d253543686174253742785f692537442925354532253744)

你可以使用sklearn的[mean\_square\_error](http://scikit-learn.org/stable/modules/generated/sklearn.metrics.mean_squared_error.html) (MSE)函数，其中，RMSE仅仅是MSE的平方根。要了解更多不同的评价指标，你可以看看[这篇文章](http://research.microsoft.com/pubs/115396/EvaluationMetrics.TR.pdf)。

由于你只是想要考虑测试数据集中的预测评分，因此，使用prediction[ground\_truth.nonzero()]筛选出预测矩阵中的所有其他元素。

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | from sklearn.metrics import mean\_squared\_error      from math import sqrt      def rmse(prediction, ground\_truth):          prediction = prediction[ground\_truth.nonzero()].flatten()          ground\_truth = ground\_truth[ground\_truth.nonzero()].flatten()  **return** sqrt(mean\_squared\_error(prediction, ground\_truth)) |

|  |  |
| --- | --- |
| 1  2 | print 'User-based CF RMSE: ' + str(rmse(user\_prediction, test\_data\_matrix))      print 'Item-based CF RMSE: ' + str(rmse(item\_prediction, test\_data\_matrix)) |

|  |  |
| --- | --- |
| 1  2 | User-based CF RMSE: 3.12663244886      Item-based CF RMSE: 3.45286971684 |

基于内存的算法事很容易实现并产生合理的预测质量的。

基于内存的CF的缺点是，它不能扩展到真实世界的场景，并且没有解决众所周知的冷启动问题，也就是当新用户或新产品进入系统时。基于模型的CF方法是可扩展的，并且可以比基于内存的模型处理更高的稀疏度，但当没有任何评分的用户或产品进入系统时，也是苦不堪言的。

# **2.基于模型的协同过滤**

基于模型的协同过滤是基于****矩阵分解（MF）****，它已获得更大的曝光，它主要是作为潜变量分解和降维的一个无监督学习方法。矩阵分解广泛用于推荐系统，其中，它比基于内存的CF可以更好地处理与扩展性和稀疏性. MF的目标是从已知的评分中学习用户的潜在喜好和产品的潜在属性（学习描述评分特征的特征），随后通过用户和产品的潜在特征的点积预测未知的评分。

当你有一个非常稀疏的多维矩阵时，通过进行矩阵分解可以调整用户-产品矩阵为低等级的结构，然后你可以通过两个低秩矩阵（其中，每行包含该本征矢量）的乘积来代表该矩阵。你通过将低秩矩阵相乘，在原始矩阵填补缺少项，以调整这个矩阵，从而尽可能的近似原始矩阵。

让我们计算MovieLens数据集的稀疏度：

|  |  |
| --- | --- |
| 1  2 | sparsity=round(1.0-len(df)/**float**(n\_users\*n\_items),3)      print 'The sparsity level of MovieLens100K is ' +  str(sparsity\*100) + '%' |

|  |  |
| --- | --- |
| 1 | The sparsity level of MovieLens100K **is** 93.7% |

举例说明用户和产品的学习隐藏偏好：假设MovieLens数据集中有以下信息：(user id, age, location, gender, movie id, director, actor, language, year, rating)。通过应用矩阵分解，模型学习到重要的用户特征是年龄组（10岁以下，10-18岁，18-30岁，30-90岁），位置和性别，而对于电影特性，它学习到年份，导演和演员是最重要的。现在，如果你看看你所存储的信息，其中并没有年份这样的特性，但该模型可以自己学习。重要方面是，CF模型仅使用数据（user\_id, movie\_id, rating）来学习潜在特征。如果只有少数可用的数据，那么基于模型的CF模式将预测不良，因为这将更难以学习潜在特征。

同时使用评分和内容特性的模型称为****混合推荐系统****，其中，协同过滤和基于内容的模型相结合。混合推荐系统通常比协同过滤或基于内容的模型自身表现出更高的精度：它们有能力更好的解决冷启动问题，因为如果你没有一个用户或者一个产品的评分，那么你可以使用该用户或产品的元数据进行预测。混合推荐系统将在未来的教程中介绍。

## 2.1SVD

一个众所周知的矩阵分解方法是****奇异值分解(SVD)****。通过使用奇异值分解，协同过滤可以被近似一个矩阵X所制定。Netflix Prize比赛中的获胜队伍使用SVD矩阵分解模型来生成产品建议，更多的信息，推荐阅读文章：[Netflix推荐：5星之外](http://techblog.netflix.com/2012/04/netflix-recommendations-beyond-5-stars.html)和[Netflix Prize和SVD](http://buzzard.ups.edu/courses/2014spring/420projects/math420-UPS-spring-2014-gower-netflix-SVD.pdf)。

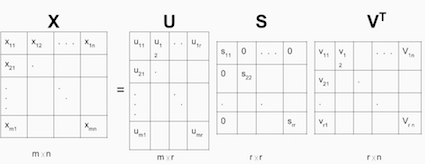
一般的方程可以表示为：

[IMG_266](https://camo.githubusercontent.com/8fef6a7b6d15069ee36471e9c6ddad5db5487a49/68747470733a2f2f6c617465782e636f6465636f67732e636f6d2f6769662e6c617465783f583d55535625354554)

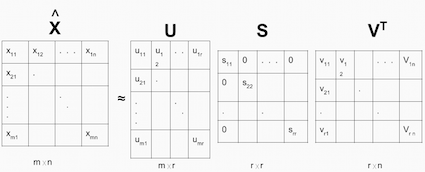
给定m x n矩阵X：

* U是一个(m x r)正交矩阵
* S是一个对角线上为非负实数的(r x r)对角矩阵
* V^T是一个(r x n)正交矩阵

S的对角线上的元素被称为X的奇异值。

矩阵X可以被分解成U，S和V。U矩阵表示对应于隐藏特性空间中的用户的特性矩阵，而V矩阵表示对应于隐藏特性空间中的产品的特性矩阵。[](https://camo.githubusercontent.com/fac165339cad3fcab544856e554a1c1ce633a0c6/687474703a2f2f7333332e706f7374696d672e6f72672f6b77677362356731622f424c4f475f4343415f352e706e67)

现在，你可以通过U, S和V^T的点积进行预测了。

[](https://camo.githubusercontent.com/481abc356614da2f29d453913e29f0b756f74222/687474703a2f2f7333332e706f7374696d672e6f72672f6368396c636d3670622f424c4f475f4343415f342e706e67)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | import scipy.sparse **as** sp      from scipy.sparse.linalg import svds        #get SVD components from train matrix. Choose k.      u, s, vt = svds(train\_data\_matrix, k = 20)      s\_diag\_matrix=np.diag(s)      X\_pred = np.dot(np.dot(u, s\_diag\_matrix), vt)        print 'User-based CF MSE: ' + str(rmse(X\_pred, test\_data\_matrix)) |

|  |  |
| --- | --- |
| 1 | User-based CF MSE: 2.72035726617 |

解决只有相对较少为人所知的问题是非常容易出现的过度拟合。SVD可能会非常缓慢，并且计算成本比较高。更近期的工作通过应用交替最小二乘或随机梯度下降最小化平方误差，并使用正则项以防止过 度拟合。你可以看到我们之前的一个关于随机梯度下降的[教程](http://online.cambridgecoding.com/notebooks/eWReNYcAfB/implementing-logistic-regression-classifier-trained-by-gradient-descent-4)，以获取更多详细信息。用于CF的交替最小二乘和随机梯度下降的方法将在未来的教程中介绍。

# 3.总结一下：

* 在这讲了如何实现简单的****协同过滤****方法，包括基于内存的CF和基于模型的CF。
* ****基于内存的模型****是基于产品或用户之间的相似性，这里使用余弦相似性。
* ****基于模型的CF****是基于矩阵分解，其中，我们使用SVD来分解矩阵。
* 构建在冷启动的情况下（其中，对于新用户和新产品来说，数据不可用）表现良好的推荐系统仍然是一个挑战。标准的协同过滤方法在这样的设置下表现不佳。