推荐系统两个最普遍的类型是基于内容过滤法和协同过滤法。协同过滤法基于用户对商品的评价信息来产生推荐，是用大众的智慧来推荐内容。相比之下，基于内容推荐系统关注的是商品的属性，基于它们之间的相似度给你推荐。

　　大体上，协同过滤（CF）是推荐引擎工作的是主力。这一算法有一个很有趣的特点，它能够自主学习，这意味着它能够开始学习哪些特点能为己所用。协同过滤又可分为基于内存的协同过滤和基于模型的协同过滤。在此教程中，你将实现运用奇异值分解的基于模型协同过滤法和通过计算余弦相似性的基于内存协同过滤法。

　　我们将使用MovieLens数据集，这是用来实现和测试推荐引擎最普遍的数据集之一。它包含来自于943个用户的10万个电影评价和1682部电影集合。你最好将这个数据集（[MoviesLens-100k](http://files.grouplens.org/datasets/movielens/ml-100k.zip)）解压到你的notebook目录。

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

**u.data**文件中包含了完整数据集。你可以在[这里](http://files.grouplens.org/datasets/movielens/ml-100k-README.txt)查阅关于这个数据集的简要描述

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

　先看一眼数据集中的前两行。接下来，让我们统计其中的用户和电影总数。

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)

Number of users = 943 | Number of movies = 1682

你可以使用[scikit-learn](http://scikit-learn.org/stable/)文库将数据集分为测试和训练两部分。[Cross\_validation.train\_test\_split](http://scikit-learn.org/stable/modules/cross_validation.html) 模块根据测试样本的百分比将数据混合并分为两部分，在这里百分比为0.25

from sklearn import cross\_validation as cv

train\_data, test\_data = cv.train\_test\_split(df, test\_size=0.25)

**基于内存协同过滤法**

　　基于内存协同过滤法可以被主要分为两部分：**用户-项目过滤（user-item filtering）**和**项目-项目过滤（ item-item filtering）**。*user-item filtering*选取一个特定用户，基于评价相似性找到与该用户相似的其他用户，并推荐那些相似用户所喜欢的项目。相比之下， *item-item filtering*先选取一个项目，然后找出也喜欢这个项目的其他用户，并找出这些用户或相似用户也喜欢的其他项目，推荐过程需要项目并输出其他项目。

* *Item-Item Collaborative Filtering*: “Users who liked this item also liked …”
* *User-Item Collaborative Filtering*: “Users who are similar to you also liked …”

在这两种情况中，你根据整个数据集创建了一个用户-项目的矩阵。因为已经把数据分成了测试和训练两部分所以你需要创建两个[943 x 1682]矩阵。训练矩阵包含75%的评价，测试矩阵包含25%的矩阵。

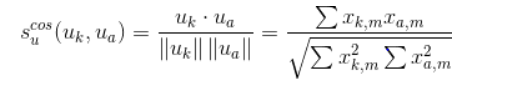
创建了用户-项目矩阵之后，计算相似性并创建一个相似度矩阵。

*Item-Item Collaborative Filtering*算法中项目之间的相似度依靠观测所有的已对相同项目评价的用户来测算。

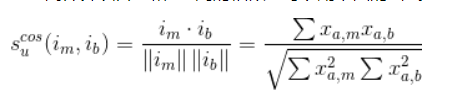
对于*User-Item Collaborative Filtering*算法，用户之间的相似性依靠观测相同用户已评价的所有项目。

　　推荐系统中通常使用余弦相似性作为距离度量，在n维孔空间中评价被视为向量，基于这些向量之间的夹角来计算相似性。

　　用户*a*和*m*可以用下面的公式计算余弦相似性，其中你可以使用用户向量*uk*和*ua*之间的点积然后除以这两个向量欧式长度之乘。



　　而计算项目m和b之间的相似度可以用下面的公式：



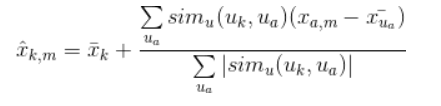
首先创建user-item矩阵，因此你需要创建两个矩阵为测试和训练数据集。

|  |  |
| --- | --- |
| 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，因为评价都为正值输出取值应为0到1.

|  |  |
| --- | --- |
| 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') |

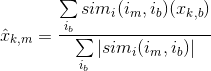
　　下一步是作出预测。既然构造了相似度矩阵*user\_similarity*和*item\_similarity，*因此你可以运用下面的公式为user-based CF做一个预测：



　　用户k和用户a之间的相似度根据一个相似用户a的一系列评价的乘积（修正为该用户的平均评价）的权重。你将需要标准化相似度这样可以使评价维持在1到5之间，最后一步，统计你想预测用户平均评价的总和。

　　这里考虑到的问题是一些用户评价所有电影时可能要么给最高分，要么给最低分。这些用户给出评价的相对不同比绝对值更重要。例如：设想，用户k对他最喜欢的电影评价4颗星，其他的好电影则评价3颗星。假设现在另一个用户t对他/她喜欢的一部电影评价为5颗星，看了想睡觉的一部电影评价为3颗星。这两位用户电影口味可能很相似但使用评价体系的方法不同。

　　当为item-based CF做一个推荐时候，你不要纠正用户的平均评价，因为用户本身用查询来做预测。



|  |  |
| --- | --- |
| 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 |

item\_prediction = predict(train\_data\_matrix, item\_similarity, type='item')

user\_prediction = predict(train\_data\_matrix, user\_similarity, type='user')

**评估**

有许多的评价指标，但是用于评估预测精度最流行的指标之一是*Root Mean Squared Error (RMSE)。*

*https://latex.codecogs.com/gif.latex?RMSE&space;=%5Csqrt%7B%5Cfrac%7B1%7D%7BN%7D&space;%5Csum&space;(x_i&space;-%5Chat%7Bx_i%7D)%5E2%7D*

你可以用sklearn中的 [mean\_square\_error](http://scikit-learn.org/stable/modules/generated/sklearn.metrics.mean_squared_error.html) (MSE)函数，RMSE只是MSE其中的一个平方根。想阅读更多关于不同评估指标你可以 查看这篇文章。

因为你仅仅想考虑在这个测试数据集中的预测评价，你可以用prediction[ground\_truth.nonzero()]过滤测试矩阵中所有其他的元素。

|  |  |
| --- | --- |
| 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: 3.1236202241  Item-based CF RMSE: 3.44983070639 |

 Memory-based算法容易实施并产生合理的预测质量。memory-based CF的缺点是它不能扩展到现实世界的场景和没有处理众所周知的冷启动问题（面对新用户或者新项目进去系统时）。Model-based CF方法可伸缩并且能处理 比memory-based方法更高等级的稀疏度，面对新用户或者没有任何评价的新项目进入系统时也会变差。我想感谢Ethan Rosenthal关于Memory-Based Collaborative Filtering的[博客](http://blog.ethanrosenthal.com/2015/11/02/intro-to-collaborative-filtering/)。

**Model-based Collaborative Filtering**

基于模型的协同过滤是基于已收到更大的曝光，主要是作为潜变量分解和降维无监督的学习方法矩阵分解（MF）

Model-based Collaborative Filtering基于已收到很多曝光的矩阵因式分解 （MF），主要是作为潜在变量分解和降维无监督学习方法。矩阵因式分解因其能比Memory-based CF更好解决可扩展性和稀疏问题而被广泛用于推荐系统。MF的目标是学习用户潜在的喜好和从已知评分的项目的潜在属性（学习描述评分特征的特点），以随后通过用户和项目的潜在特征点积预测未知的评分。

当你有一个多维度稀疏矩阵，通过矩阵因式分解你能够将用户-项目矩阵（user-item matrix）重构成低评分结构（low-rank structure），并且你能够通过两个低评分（ low-rank）矩阵相乘得出这个矩阵，其中矩阵的行包含潜在向量。

 通过低评价矩阵乘积尽可能调整这个矩阵近似原始矩阵，以填充原始矩阵中缺失的项。

现在开始计算MovieLens 数据集的稀疏等级：

|  |  |
| --- | --- |
| 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% |

　　举个例子说明学习用户和项目的潜在喜好：就拿MovieLens数据集来说你有一下信息：（用户ID，年龄，地理位置，性别，电影ID，导演，演员，语言，年份，评分）。通过运用矩阵因式分解这个模型学习到重要的用户特征是年龄段（10岁以下、10-18岁、18-30岁、30-90岁）、地理位置和 性别，对电影特征它学习到最重要的是年代、导演和演员。现在如果你回过头来看你已经存储的信息，没有诸如年代等的特征，但是这个模型可以自己学习到。重要的方面是CF模型仅需使用数据（用户ID，电影ID，评分）来学习这些潜在的特征。如果没有数据可用CF模型性能将会很差，因为这样它更困难学习这些潜在的特征。

评分和特征都要使用的模型称为混合推荐系统（**Hybrid Recommender Systems**），它是Collaborative Filtering 和Content-based 模型两者的结合。混合推荐系统通常比Collaborative Filtering 或者Content-based模型独立地表现出更高的精度：它们能够更好地处理冷启动问题（因为如果你没有可用于数据集的对于用户或项目的任何评价你就很难做出预测）。混合推荐系统将在下一篇介绍中介绍。

**[SVD](http://online.cambridgecoding.com/notebooks/eWReNYcAfB/implementing-your-own-recommender-systems-in-python-2" \l "svd)**

 一个知名的矩阵因式分解方法是**Singular value decomposition (SVD)。**Collaborative Filtering **协同过滤可以通过使用奇异值分解近似矩阵X被配制**