从零开始用 python 搭建推荐引擎

数据派

(给Python开发者加星标，提升Python技能)

作者：Pulkit Sharma；翻译：申利彬，付宇帅

简介

当今社会的每个人都面临着各种各样的选择。例如，如果我漫无目的想找一本书读，那么关于我如何搜索就会出现很多可能。这样一来，我可能会浪费很多时间在网上浏览，并且在各种各样的网站上搜寻，希望能找到有价值的书籍。这个时候我可能寻找别人的推荐。

如果有一家网站或者手机应用可以基于我以前阅读的书籍向我推荐新的书籍，那对我肯定有很大的帮助。这时我会有如下愉快的体验，登录网站，就可以看到符合我兴趣的10本书籍，不用浪费时间在网站上搜寻。

这就是推荐引擎所做的事情，它们的力量现在正被大多数企业所使用。从亚马逊到Netflix，谷歌到谷歌阅读，推荐引擎是机器学习技术中最广泛的应用之一。

在本文中，将介绍各种推荐引擎算法以及使用Python构建它们的基本框架。我们还将讨论这些算法工作背后的数学原理，最后使用矩阵分解技术创建属于我们自己的推荐引擎。

1 什么是推荐引擎?

一直到现在，人们也会倾向于买朋友或者信任的人推荐商品。当对某个商品有任何疑问时，人们往往会采用这种方式。但是随着数字时代的到来，这个圈子已经扩展到包括使用某种推荐引擎的在线网站。

一个推荐引擎使用不同的算法过滤数据，并向用户推荐最相关的物品。它首先存储客户过去的行为数据，然后基于这些数据向客户推荐他们可能购买的物品。

如果一个全新的用户访问一个电子商务网站，网站没有该用户的任何历史数据。那么在这样的场景中，网站是如何向用户推荐产品呢？一种可能的方法是向客户推荐卖的最好的商品，也就是该商品需求量很大。还有另外一种可能的方法是向用户推荐可以给网站带来最大利润的商品。

如果我们可以根据用户的需要和兴趣向用户推荐一些商品，这可以对用户体验产生积极的影响，最后可以达到多次访问的效果。因此，现在的企业通过研究用户过去的行为数据来构建聪明和智能的推荐引擎。

目前我们对推荐引擎有了直观的认识，现在让我们来看看它们是如何工作的。

2 推荐引擎是如何工作的?

在深入探讨这个主题之前，我们首先考虑一下如何向用户推荐商品：

- 我们可以向一个用户推荐最受欢迎的商品
- 可以根据用户偏好（用户特征）把用户分为多个细分类别，然后基于他们属于的类别推荐商品。

上述两种方法都有缺点。在第一种方法中，对于每一个用户来说最受欢迎的商品都是相同的，所以用户看到的推荐也是相同的。在第二种方法中，随着用户数量的增加，用户特征也随着增加。因此将用户划分为多个类别将会是一件非常困难的任务。

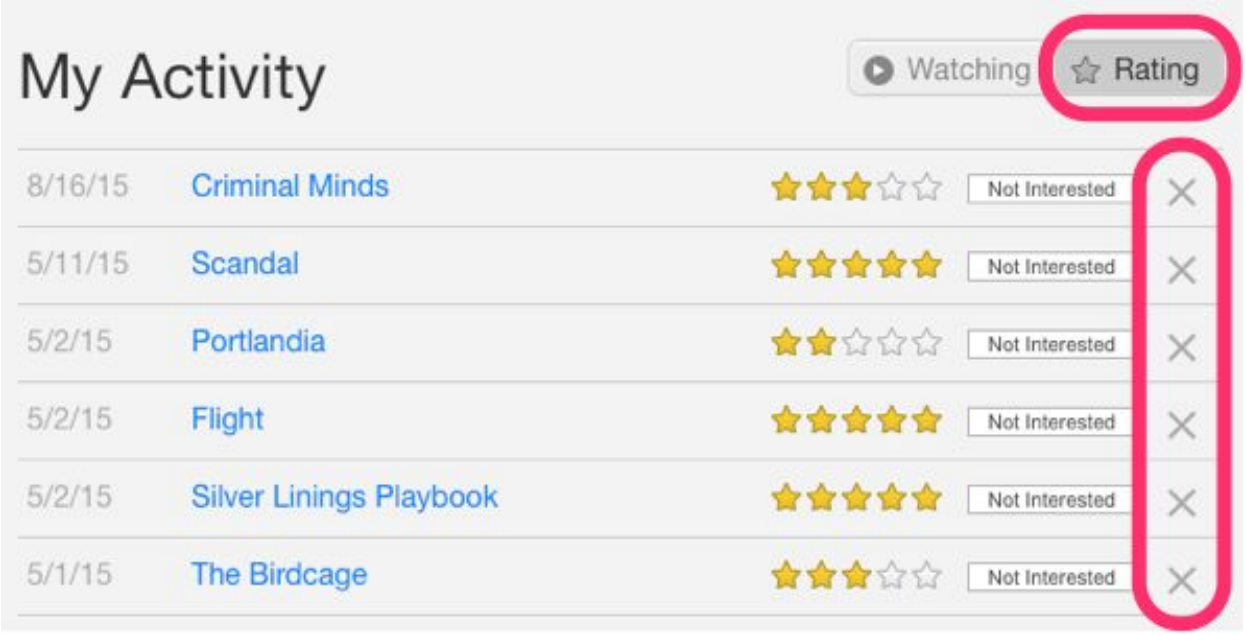
这里的主要问题是我们无法为用户具体的兴趣定制推荐。这就像亚马逊建议你买一台笔记本电脑，仅仅是因为它被大多数购物者购买。但幸运的是，亚马逊（或其他大公

司）并没有使用上述方法来推荐商品。他们使用一些个性化的方法来帮助他们更准确地推荐产品。

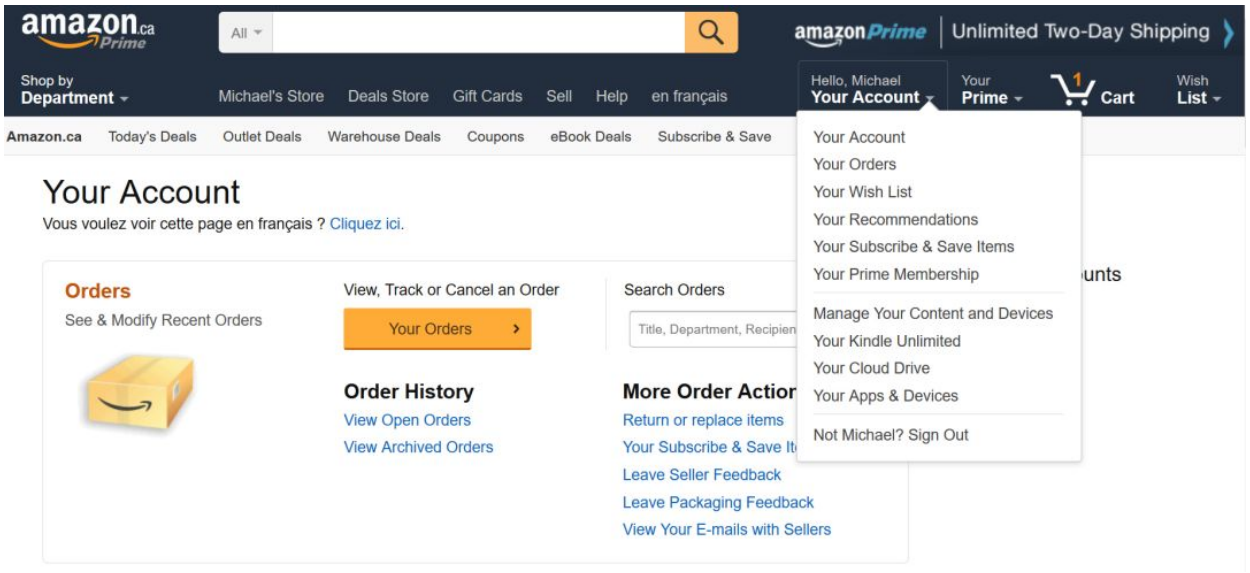
我们现在来看看推荐引擎是如何通过以下步骤来工作的。

2.1 数据收集

收集数据是构建推荐引擎的第一步也是最关键的一步。可以通过两种方式收集数据：显式和隐式。显示数据是用户有意提供的信息，比如电影排名，相反隐氏数据则不是用户主动提供，而是从数据流中收集得到的信息，例如搜索历史、点击率、历史订单等。



在上面的图片中，Netflix正在以用户对不同电影的评分形式明确地收集数据。



上图可以看到Amazon记录的用户历史订单，这是一个隐式数据收集模式的例子。

2.2 数据存储

数据量决定了模型的建议有多好，例如，在电影推荐系统中，用户对电影的评价越多，推荐给其他用户的效果就越好。数据类型对采用何种存储类型有很重要的影响，这种类型的存储可以包括一个标准的SQL数据库、NoSQL数据库或某种类型的对象存储。

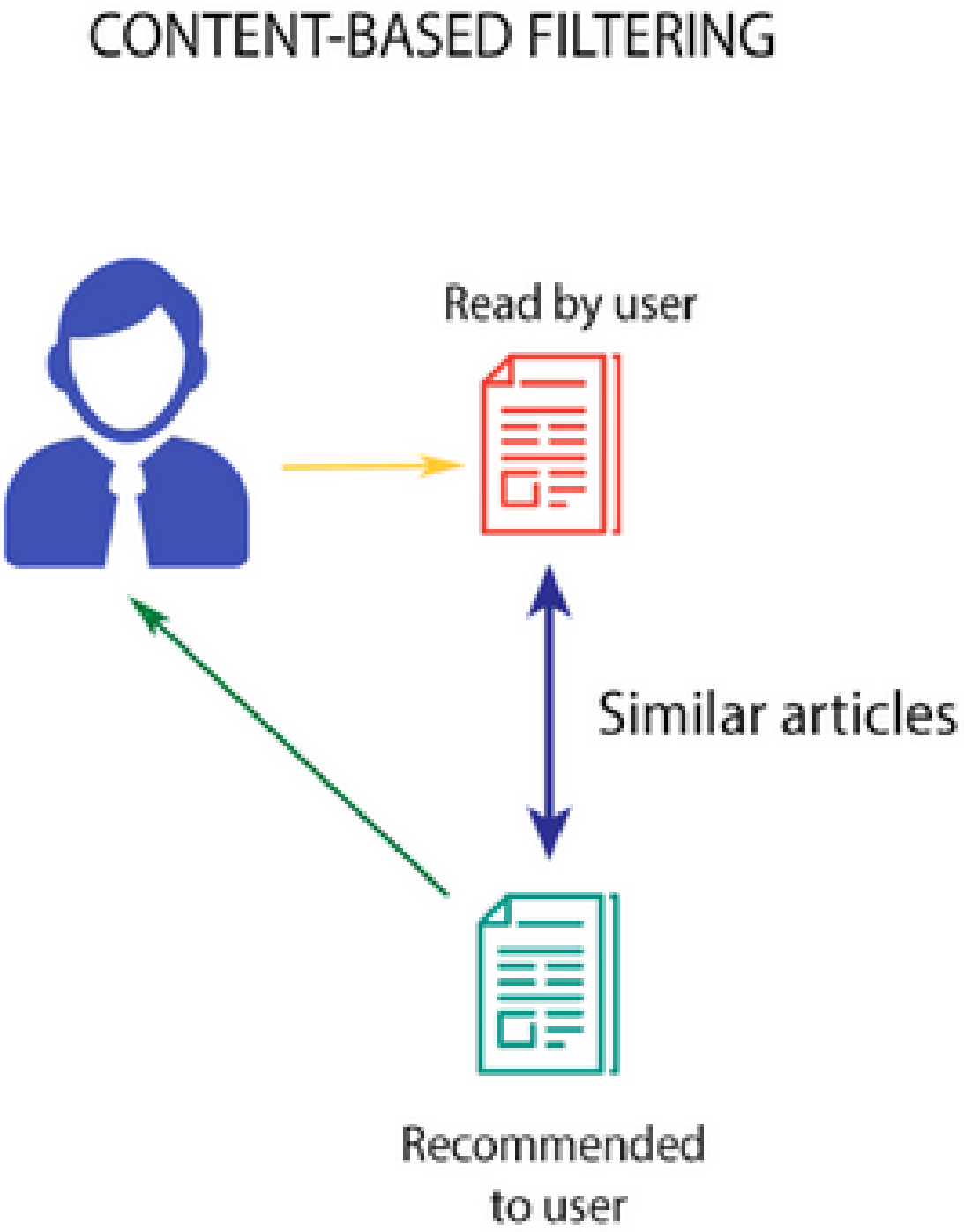
2.3 数据过滤

在收集和存储数据之后，我们必须对其进行过滤，以便提取出最终推荐所需的相关信息。

有各种各样的算法可以帮助我们简化过滤过程。在下一节中，我们将详细介绍每种算法。

2.3.1 基于内容的过滤

这个算法推荐的产品类似于用户过去喜欢的产品。



图片来源：Medium

例如，如果一位用户喜欢《盗梦空间》这部电影，那么算法就会推荐属于同一类型的电影。但是，算法是如何理解选择和推荐电影的类型呢？

以Netflix为例：它们以向量形式保存与每个用户相关的所有信息。这个向量包含用户过去的行为，也就是用户喜欢/不喜欢的电影和他们给出的评分，这个向量也被称为

轮廓向量（profile vector）。所有与电影相关的信息都存储在另一个叫做项目向量（item vector）中。项目向量包含每个电影的细节，如类型、演员、导演等。

基于内容的过滤算法找到了轮廓向量与项目向量夹角的余弦，也就是余弦相似度。假设A是轮廓矢量，B是项目矢量，那么它们之间的相似性可以按如下公式计算：

$$sim(A,B) = \cos(\theta) = \frac{A \cdot B}{\|A\|\|B\|}$$

根据在-1到1之间的余弦值，可以将电影按降序排列，并且采用下面两种方法中的一种用于推荐：

- 选择前N部电影：推荐最相关的前N部电影（这里N可以由公司决定）。
- 等级量表的方法：设置一个阈值，并推荐所有超过该阈值的电影。

其它可以用来计算相似性的方法有：

- 欧几里得距离：如果在N维空间中绘制，相似的实体将会彼此靠近。因此，我们可以计算实体之间的距离根据这个距离，向用户推荐内容。下面是欧几里得距离公式：

$$\text{Euclidean Distance} = \sqrt{(x_1 - y_1)^2 + \ldots + (x_N - y_N)^2}$$

- 皮尔逊相关性：它告诉我们两个实体的相关程度，越高的相关性，就越相似。皮尔逊的相关性可以用以下公式来计算：

$$sim(u,v) = \frac{\sum(r_{ui} - \bar{r}_u)(r_{vi} - \bar{r}_v)}{\sqrt{\sum(r_{ui} - \bar{r}_u)^2}\sqrt{\sum(r_{vi} - \bar{r}_v)^2}}$$

这种算法有一个主要的缺点，也就是它仅限于推荐相同类型的实体。它永远不会推荐用户过去没有购买或喜欢的产品。因此，如果用户过去仅仅看或喜欢动作电影，系统也就只会推荐动作电影。很显然，这种搭建推荐引擎的方法泛化性能很差。

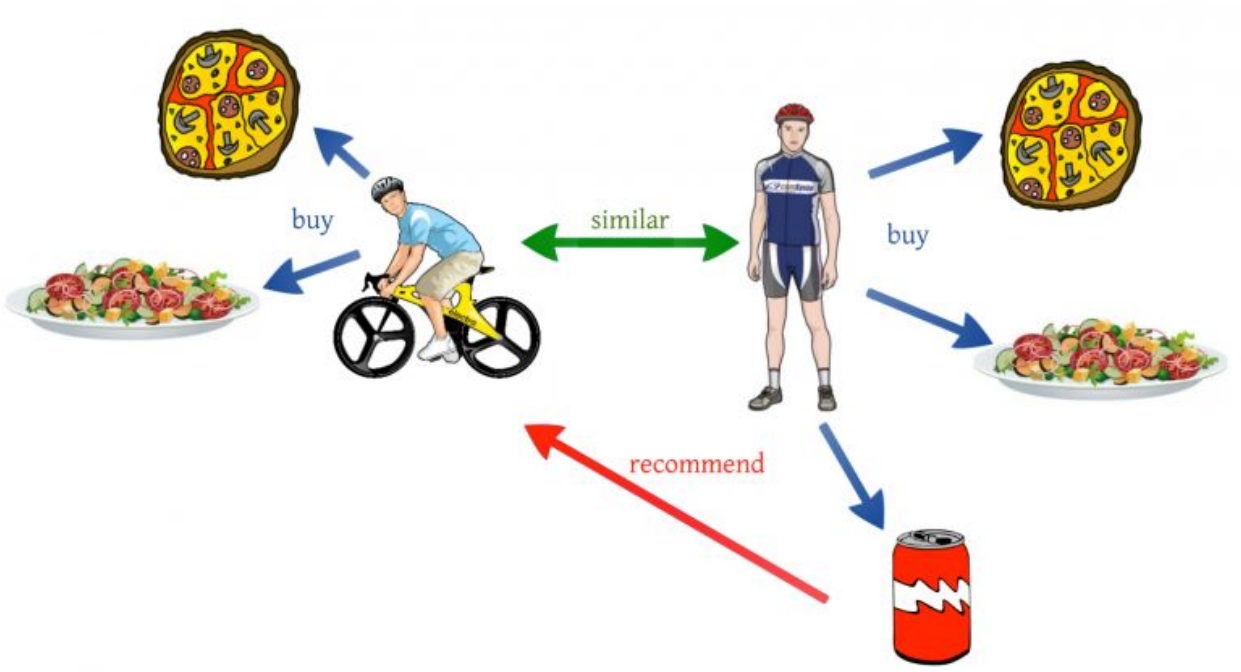
我们要找到一种算法可以改进这种类型的推荐系统，它不仅可以根据内容进行推荐，还要可以利用用户的行为信息。

2.3.2 协同过滤

我们通过一个例子来理解这个方法。如果用户A喜欢3部电影，比如《星际穿越》，《盗梦空间》和《前目的地》，而用户B喜欢《盗梦空间》、《前目的地》和《致命魔术》，那么他们就有差不多的兴趣爱好。我们可以肯定地说，A应该喜欢《致命魔术》，B应该喜欢《星际穿越》。协同过滤算法使用“用户行为”来推荐电影。这是工业中最常用的算法之一，因为它不依赖于任何额外的信息。协同过滤技术有很多种不同的类型，我们将在下面详细讨论这些问题。

用户与用户间的协同过滤

该算法首先发现用户之间的相似性分数，基于这个相似性的分数，它会挑选出最相似的用户，并推荐这些类似的用户以前喜欢或购买的产品。



图片来源：Medium

就我们之前的电影例子而言，这个算法根据他们之前给不同电影的评分来发现每个用户之间的相似性。用户u的一个实体的预测是通过计算其它用户对一个实体i的用户评分的加权总和来计算的。Pu，i通过下式计算得到：

$$P_{u,i} = \frac{\sum_v (r_{v,i} * s_{u,v})}{\sum_v s_{u,v}}$$

公式符号含义如下：

- Pu，I 是一个实体的预测
- Rv，I 是用户v对电影i的评分
- Su，v 使用户之间的相似性分数

现在，我们在轮廓向量中对用户进行了评分，并且基于这个向量，我们要预测其他用户的评分。接下来的步骤如下：

- 对于预测，我们需要用户u和v之间的相似性。这时可以利用皮尔逊相关性。
- 首先，我们发现被用户打分的商品，根据评分，计算用户之间的相关性。

- 可以用相似值来计算预测。这个算法首先计算每个用户之间的相似性，然后根据每个相似度计算预测值。具有高相关性的用户，一般都相似。
- 基于这些预测值给出推荐。我们通过一个例子来理解它：

用户-电影评分矩阵：

User/Movie	x1	x2	x3	x4	x5	Mean User Rating
A	4	1	–	4	–	3
B	–	4	–	2	3	3
C	–	1	–	4	4	3

我们可以看到一个用户-电影评分矩阵，为了更深入地理解这个公式，让我们在上表中找到用户(A, C)和(B, C)之间的相似性。A与C共同评分的电影是x2和x4，B与C共同评分的电影是x2,x4和x5。

$$r_{AC} = [(1-3)*(1-3) + (4-3)*(4-3)]/[((1-3)^2 + (4-3)^2)^{1/2} * ((1-3)^2 + (4-3)^2)^{1/2}] = 1$$

$$r_{BC} = [(4-3)*(1-3) + (2-3)*(4-3) + (3-3)*(4-3)]/[((4-3)^2 + (2-3)^2 + (3-3)^2)^{1/2} * ((1-3)^2+(4-3)^2 + (4-3)^2)^{1/2}] = -0.866$$

用户A和C之间的相关性大于B和C之间的相关性。因此用户A和C有更多的相似性，用户A喜欢的电影会推荐给用户C，反之亦然。

这个算法非常耗时，因为它涉及到计算每个用户的相似度，然后计算每个相似度得分的预测。解决这个问题的一种方法是只选择几个用户(邻居)而不是对所有的值进行预测，也就是说，我们只选择几个相似值而不是对所有相似值进行预测：

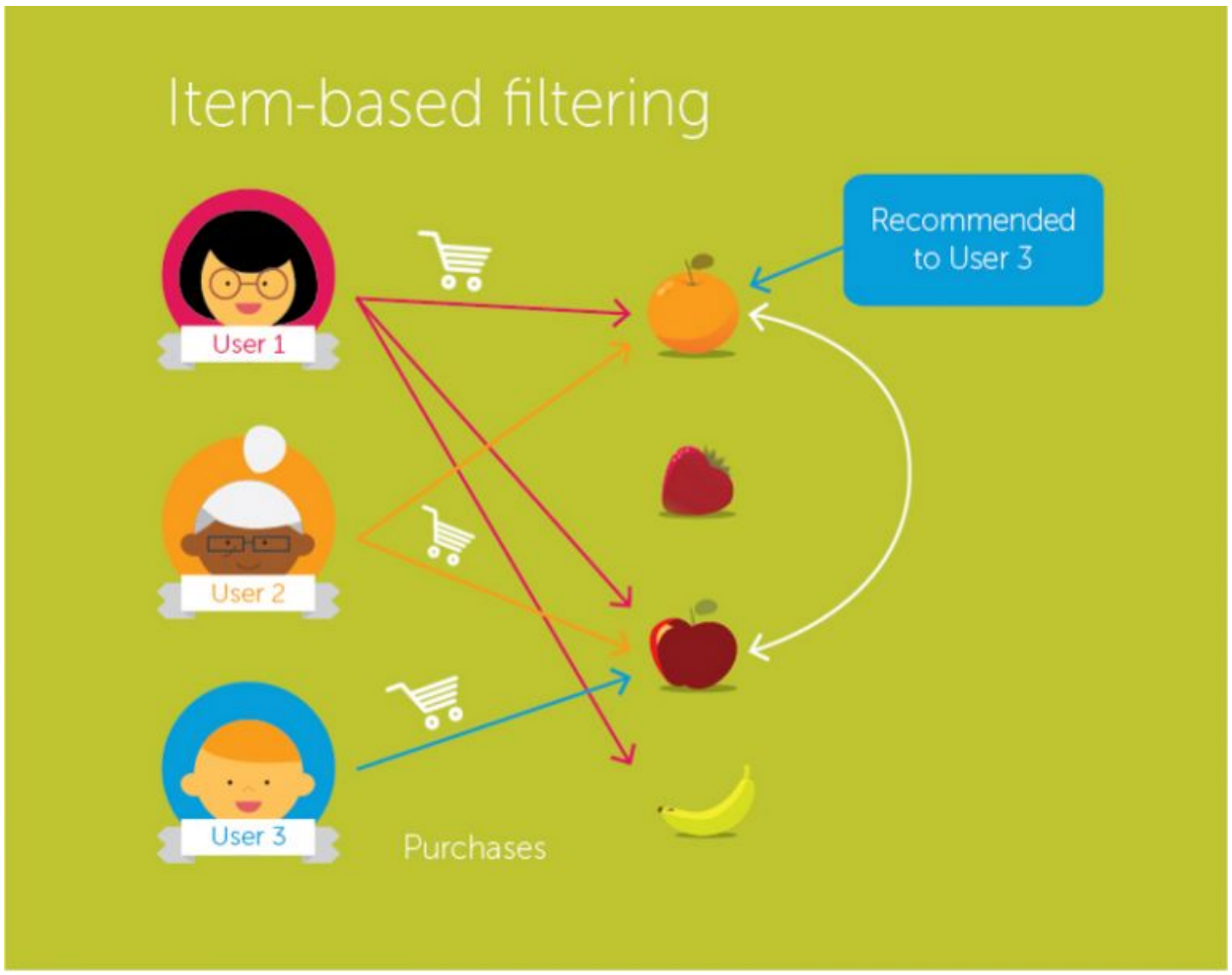
- 选择一个相似度阈值并选择该值以上的所有用户
- 随机选择用户
- 按照相似度值的降序排列相邻用户，然后选择前n个用户

- 使用聚类算法选择相邻用户

当用户数量较少时，这个算法可以很好的发挥作用。当有大量的用户时，它并不有效，因为计算所有用户对之间的相似性需要花费大量的时间。这就产生了商品-商品的协同过滤，当用户数量远远超过推荐商品的数量时，这种算法是非常有效的。

商品-商品协同过滤

在这个算法中，我们计算每一对商品之间的相似度。



图片来源：Medium

所以在我们的案例中，我们会发现每个电影对之间的相似性，在此基础上，我们可以推荐用户过去喜欢的相似的电影。这个算法的工作原理类似于用户-用户协同过滤，仅仅做了一点小小的改变——不是对“相邻用户”的评分进行加权求和，而是对“相邻商品”的评分进行加权求和。预测公式如下：

$$P_{u,i} = \frac{\sum_N (s_{i,N} * R_{u,N})}{\sum_N (|s_{i,N}|)}$$

我们计算商品之间的相似性：

$$sim(i,j) = cos(\vec{i}, \vec{j}) = \frac{\vec{i} \cdot \vec{j}}{\|\vec{i}\|_2 * \|\vec{j}\|_2}$$

现在我们我们有每一对电影的相似性，评分和预测都已经有了，而且基于这些预测，我们可以进行相似电影的推荐。我们通过一个例子来理解：

User/Movie	x1	x2	x3	x4	x5
A	4	1	2	4	4
B	2	4	4	2	1
C	–	1	–	3	4
Mean Item Rating	3	2	3	3	3

这里的电影评分均值是所有个某一特定电影评分的平均值（将它与我们在用户-用户过滤中看到的表进行比较）。并且我们不是像前面看到的那样找到用户-用户相似度，而是找到商品-商品相似度。

要做到这一点，首先我们需要找到对这些商品进行评分的用户，并根据评分计算商品之间的相似性。我们来找出电影(x1, x4)和(x1, x5)之间的相似性。从上表可以看出，给电影x1, x4都有打分的用户是A与B，给电影x1, x5都有打分的用户也是A与B。

电影x1和x4的相似度大于电影x1和x5的相似度，基于这些相似度值，如果有任何用户搜索电影 x1，那么将会为他们推荐电影x4，反之也一样。在进一步运用这些概念之前，有一个问题我们必须要知道答案——如果在数据集中添加了新用户或新电影，将会发生什么？这被称为冷启动，它有两种类型:

- 用户冷启动
- 产品冷启动

用户冷启动意味着数据库中新增加了一位新用户，由于没有该用户的历史记录，系统也就不知道该用户的偏好，所以向这位用户推荐商品就会很困难。所以我们将如何解决这个问题呢？一种基本的方法是采用基于流行度的策略，即推荐最受欢迎的产品。这些可以由最近的流行趋势来决定，以后一旦我们知道了用户的喜好，推荐商品就会变的很容易。

另一方面，产品冷启动意味着新产品投放市场或添加到系统中。用户的行为对决定任何产品价值来说都是很重要的。产品接受的交互越多，我们的模型就越容易向正确的用户推荐该产品。我们可以利用基于内容的过滤来解决这个问题。系统首先使用新产品的内容进行推荐，不过最终使用用户对该产品的交互进行推荐。

现在让我们使用Python中的一个案例学习来巩固我们对这些概念的理解。这个例子非常有趣，快打开你的电脑准备开始吧。

3 基于MovieLens数据集的python实例学习

我们将使用这个MovieLens数据集建立一个模型并向最终用户推荐电影。明尼苏达大学（University of Minnesota）的GroupLens研究项目已经收集了这些数据。数据集可以从这里下载：

https://grouplens.org/datasets/movielens/100k/

这个数据集包含有：

- 1682部电影中有943位观众观看了10万次(1-5次)
- 用户的人口统计信息(年龄、性别、职业等)

首先，我们将导入标准库并将数据读入python中：

```
import pandas as pd

%matplotlib inline



import matplotlib

import matplotlib.pyplot as plt

import numpy as np
```

```
# pass in column names for each CSV as the column name is not given in the file and read them using pandas.

# You can check the column names from the readme file



#Reading users file:

u_cols = ['user_id', 'age', 'sex', 'occupation', 'zip_code']

users = pd.read_csv('ml-100k/u.user', sep='|', names=u_cols,encoding='latin-1')



#Reading ratings file:

r_cols = ['user_id', 'movie_id', 'rating', 'unix_timestamp']

ratings = pd.read_csv('ml-100k/u.data', sep='\t', names=r_cols,encoding='latin-1')



#Reading items file:



i_cols = ['movie id', 'movie title' ,'release date','video release date', 'IMDb URL', 'unknown', 'Action', 'Adventure',

'Animation', 'Children\'s', 'Comedy', 'Crime', 'Documentary', 'Drama', 'Fantasy',

'Film-Noir', 'Horror', 'Musical', 'Mystery', 'Romance', 'Sci-Fi', 'Thriller', 'War', 'Western']

items = pd.read_csv('ml-100k/u.item', sep='|', names=i_cols,

encoding='latin-1')
```

加载数据集之后，我们应该查看每个文件的内容(用户、评分、电影)：

- **用户**

```
print(users.shape)
```

```
users.head()
```

```
(943, 5)
```

	user_id	age	sex	occupation	zip_code
0	1	24	M	technician	85711
1	2	53	F	other	94043
2	3	23	M	writer	32067
3	4	24	M	technician	43537
4	5	33	F	other	15213

因此，我们可以看到数据集中有943个用户，每个用户有5个特性，即用户ID、年龄、性别、职业和邮政编码。现在我们来看看评分文件。

- **评分**

```
print(ratings.shape)
```

```
ratings.head()
```

```
(100000, 4)
```

	user_id	movie_id	rating	unix_timestamp
0	196	242	3	881250949
1	186	302	3	891717742
2	22	377	1	878887116
3	244	51	2	880606923
4	166	346	1	886397596

对于不同的用户和电影组合，我们有100k个电影评分。现在最后检查电影文件。

- **电影**

```
print(items.shape)
```

```
items.head()
```

```
(1682, 24)
```

	movie id	movie title	release date	video release date	IMDb URL	unknown	Action	Adventure	Animation	Children's	...	Fantasy	Film-Noir	Horror	Musical	M
0	1	Toy Story (1995)	01-Jan-1995	NaN	http://us.imdb.com/M/title-exact?Toy%20Story%2...	0	0	0	1	1	...	0	0	0	0	
1	2	GoldenEye (1995)	01-Jan-1995	NaN	http://us.imdb.com/M/title-exact?GoldenEye%20(...	0	1	1	0	0	...	0	0	0	0	
2	3	Four Rooms (1995)	01-Jan-1995	NaN	http://us.imdb.com/M/title-exact?Four%20Rooms%...	0	0	0	0	0	...	0	0	0	0	
3	4	Get Shorty (1995)	01-Jan-1995	NaN	http://us.imdb.com/M/title-exact?Get%20Shorty%...	0	1	0	0	0	...	0	0	0	0	
4	5	Copycat (1995)	01-Jan-1995	NaN	http://us.imdb.com/M/title-exact?Copycat%20(1995)	0	0	0	0	0	...	0	0	0	0	

5 rows × 24 columns

这个数据集包含了1682部电影的属性，一共有24列，其中最后19列指定了具体电影的类型。这些是二进制列，即，值1表示该电影属于该类型，否则为0。

GroupLens已经将数据集划分为train和test，每个用户的测试数据有10个等级，总共9430行。我们接下来把这些文件读入到python环境中。

```
r_cols = ['user_id', 'movie_id', 'rating', 'unix_timestamp']

ratings_train = pd.read_csv('ml-100k/ua.base', sep='\t', names=r_cols, encoding='latin-1')

ratings_test = pd.read_csv('ml-100k/ua.test', sep='\t', names=r_cols, encoding='latin-1')

ratings_train.shape, ratings_test.shape
```

现在终于到了构建我们推荐引擎的时候了!

4 从0搭建协同过滤模型

我们将根据用户-用户相似度和电影-电影相似度推荐电影。为此，我们首先需要计算独立用户和电影的数量。

```
n_users = ratings.user_id.unique().shape[0]

n_items = ratings.movie_id.unique().shape[0]
```

现在，我们将创建一个用户电影矩阵，该矩阵可用于计算用户与电影之间的相似性。

```
data_matrix = np.zeros((n_users, n_items))

for line in ratings.itertuples():
```

```
    data_matrix[line[1]-1, line[2]-1] = line[3]
```

现在，我们来计算相似度。我们可以使用sklearn的pairwise_distance函数来计算余弦相似度。

```
from sklearn.metrics.pairwise import pairwise_distances

user_similarity = pairwise_distances(data_matrix, metric='cosine')

item_similarity = pairwise_distances(data_matrix.T, metric='cosine')
```

这就给出了数组表单中的item-item和user-user相似度。下一步是根据这些相似数值做出预测，下面我们定义一个函数来做这个预测。

```
def predict(ratings, similarity, type='user'):

    if type == 'user':

        mean_user_rating = ratings.mean(axis=1)

        #We 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
```

最后，我们将基于用户相似度和电影相似度进行预测。

```
user_prediction = predict(data_matrix, user_similarity, type='user')
```

```
item_prediction = predict(data_matrix, item_similarity, type='item')
```

事实证明，我们还有一个库，可以自动生成所有这些推荐。现在让我们学习如何在Python中使用turicreate创建推荐引擎。要熟悉turicreate并将它安装到你的电脑上，请参考这里：

https://github.com/apple/turicreate/blob/master/README.md

5 使用Turicreate搭建简单流行的协同过滤模型

在安装好turicreate库之后，首先导入它，然后在我们的环境中读取训练和测试数据集。

```
import turicreate

train_data = turicreate.SFrame(ratings_train)

test_data = turicreate.Sframe(ratings_test)
```

我们有用户行为，也有用户和电影的属性，所以我们可以制作基于内容和协同过滤算法。我们将从一个简单的流行模型开始，然后构建一个协同过滤模型。

首先，我们建立一个向用户推荐最流行电影的模型，也就是所有用户都会收到相同的推荐。我们可以使用Turicreate中的popularity_recommender推荐函数来实现。

```
popularity_model = turicreate.popularity_recommender.create(train_data,user_id='user_id', item_id='movie_id', target='rating')
```

我们使用的各种变量有：

- train_data：SFrame包含了我们所需要的训练数据
- user_id：这一列包含了每个用户的ID
- item_id: 这一列包含了每一个要被推荐的电影（电影ID）

- target:这一列包含了用户给的评分或等级

预测的时间到了!我们将为我们数据集中的前5个用户推荐排名前5的电影。

```
popularity_recomm = popularity_model.recommend(users=[1,2,3,4,5],k=5)

popularity_recomm.print_rows(num_rows=25)
```

user_id	movie_id	score	rank
1	1467	5.0	1
1	1201	5.0	2
1	1189	5.0	3
1	1122	5.0	4
1	814	5.0	5
2	1467	5.0	1
2	1201	5.0	2
2	1189	5.0	3
2	1122	5.0	4
2	814	5.0	5
3	1467	5.0	1
3	1201	5.0	2
3	1189	5.0	3
3	1122	5.0	4
3	814	5.0	5
4	1467	5.0	1
4	1201	5.0	2
4	1189	5.0	3
4	1122	5.0	4
4	814	5.0	5
5	1467	5.0	1
5	1201	5.0	2
5	1189	5.0	3
5	1122	5.0	4
5	814	5.0	5

[25 rows x 4 columns]

注意，所有用户的推荐都是一样的——1467、1201、1189、1122、814。它们的顺序是一样的!这证实了所有推荐电影的平均评分都是5分，即所有观看电影的用户都给予了最高的评分。因此我们基于流行的系统表现是符合我们预期的。

在构建了流行模型之后，我们现在将构建一个协同过滤模型。我们来训练电影相似度模型，并为前5名用户提供前5项推荐。

```
#Training the model

item_sim_model = turicreate.item_similarity_recommender.create(train_data, user_id='user_id', item_id='movie_id', target='rating', similarity_type='cosine')

#Making recommendations

item_sim_recomm = item_sim_model.recommend(users=[1,2,3,4,5],k=5)

item_sim_recomm.print_rows(num_rows=25)
```

user_id	movie_id	score	rank
1	423	0.9850328512319172	1
1	202	0.9431346880115625	2
1	655	0.7932585664377868	3
1	403	0.765623665037956	4
1	568	0.7633005562629408	5
2	50	1.1256258487701416	1
2	181	1.0651773168490484	2
2	7	0.9754774272441864	3
2	121	0.94162796323116	4
2	9	0.831989913032605	5
3	313	0.6353766620159149	1
3	328	0.6032880300825293	2
3	315	0.5422587123784152	3
3	331	0.5355071858926252	4
3	332	0.5316696112806146	5
4	50	1.1311477082116264	1
4	288	1.0487151145935059	2
4	181	0.9505999386310577	3
4	7	0.9417778807027	4
4	302	0.9139021464756557	5
5	195	1.0158377741322373	1
5	202	0.9353599468866984	2
5	56	0.8498673283692563	3
5	82	0.7769144300258521	4
5	96	0.7397452755407854	5

[25 rows x 4 columns]

在这里我们可以看到每个用户的推荐(movie id)是不同的。对于不同的用户，我们有不同的推荐集，也就是说个性化是存在的。

在这个模型中，我们没有每个用户给出的每个电影的评分。我们必须找到一种方法来预测所有这些缺失的评分。为此，我们必须找到一组可以定义用户如何评价电影的特

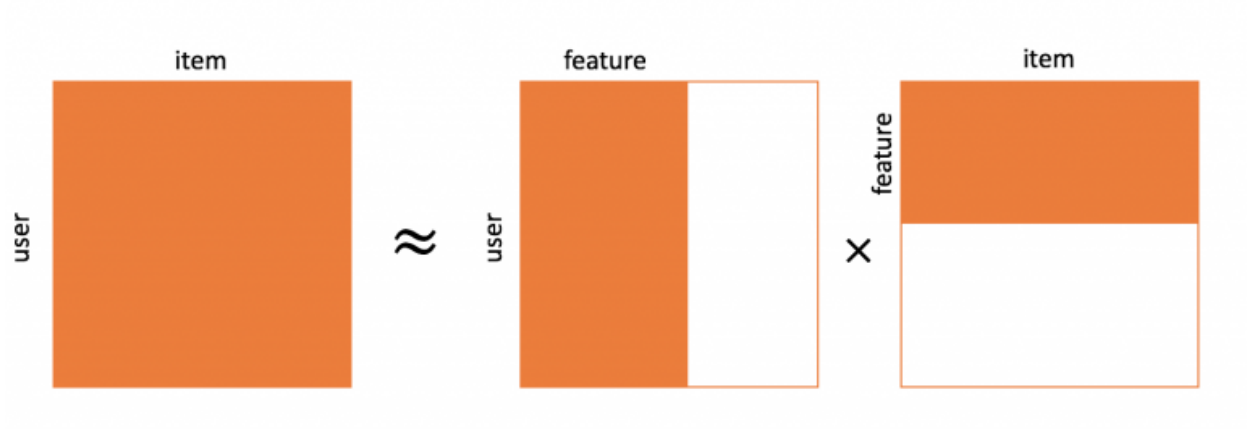
征。这些被称为潜在特征（latent features）。我们需要找到一种方法，从现有的特征中提取出最重要的潜在特征。下一节将介绍矩阵分解技术，它使用低维密集矩阵，帮助我们提取重要的潜在特征。

6 矩阵分解简介

我们通过一个例子来理解矩阵分解。考虑不同用户对不同电影给出的用户电影评分矩阵(1-5)。

movie_id	1	2	3	4	5
user_id					
1	5.0	3.0	4.0	3.0	3.0
2	4.0	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0	0.0
5	4.0	3.0	0.0	0.0	0.0

这里的用户id是不同用户的唯一id，每个电影也被分配一个唯一id。0.0表示用户没有对特定的电影进行评分(1是用户能给出的最低评分)。我们希望预测这些缺失的评分，使用矩阵分解可以找到一些潜在的特征，它们可以决定用户如何评价一部电影。我们将原矩阵分解成不同的组成部分，使这些部分的乘积等于原始矩阵。



假设我们要找到k个潜在特征。因此，我们可以将我们的评分矩阵R(MxN)划分为P(MxK)和Q(NxK)，使P x QT(这里QT是Q矩阵的转置)近似于R矩阵:

$$R=P\Sigma Q^T$$

:

- M是用户总数
- N是电影的总数
- K是总的潜在特征
- R是MxN用户电影评分矩阵
- P是MxK用户特征关联矩阵，表示用户与特征之间的关联
- Q是NxK电影特征关联矩阵，表示电影与特征之间的关联
- Σ是K*K个对角特征权重矩阵，代表了特征的重要权重

通过矩阵分解的方法来选择潜在特征并消除了数据中的噪声。如何做到的呢？它是删除了不能决定用户如何评价电影的特征。现在要得到用户puk对一部电影qik的所有潜在特征k的评分rui，我们可以计算这两个向量的点积，并将它们相加，得到基于所有潜在特征的评分。

$$r_{ui} = \Sigma^{K}_{k=1}\, p_{uk}\, \sigma_k\, q_{ki}$$

这就是矩阵分解给我们预测电影的评分，而这些电影并没有得到用户的评分。但是，我们如何将新数据添加到我们的用户电影评分矩阵中，也就是说，如果一个新用户加入并对电影进行评分，我们将如何将这些数据添加到已有的矩阵中?

我通过矩阵分解的方法让你更容易理解这个过程。如果有一个新用户进入系统，对角权重矩阵和商品-特征相关性矩阵是不会发生变化的，唯一的变化是发生在用户特征关联矩阵P中。我们可以用一些矩阵乘法来实现这个。

我们有：

$$R=P\Sigma Q^T$$

两边同时乘以矩阵Q：

$$RQ=P\Sigma Q^TQ$$

现在我们有：

$$Q^TQ = 1$$

所以：

$$RQ=P\Sigma$$

进一步化简，得到P矩阵:

$$RQ\Sigma^{-1}=P$$

这是更新后的用户特征关联矩阵。同样地，如果向系统中添加了新电影，我们可以按照类似的步骤得到更新后的电影特征关联矩阵Q。

我们要有意识，虽然把R矩阵分解成P和Q，但是我们如何决定哪个P和Q矩阵更加近似于R矩阵呢？我们可以用梯度下降算法来做这个，目标是最小化实际评分与使用P和Q进行评估的评分之间的平方误差。平方误差公式如下所示:

$$e_{ui}^2 = (r_{ui} - \check{r}_{ui})^2 = (r_{ui} - \Sigma^K_{k=1} p_{uk}\, \sigma_k\, q_{ki})^2$$

- eui 表示误差
- rui 表示用户u对电影i的实际评分
- řui 表示预测用户u对电影i的评分

我们的目标是确定p和q值，使误差最小化，因此需要更新p值和q值，以得到这些矩阵的优化值，这样误差最小。现在我们将为puk和qki定义一个更新规则，在梯度下降中的更新规则是由要最小化的误差梯度定义的。

$$\frac{\partial}{\partial puk}(e_{ui}^2) = -2(r_{ui} - \check{r}_{ui})\, q_{ki} = -2e_{ui}\, q_{ki}$$

$$\frac{\partial}{\partial qki}(e_{ui}^2) = -2(r_{ui} - \check{r}_{ui})\, p_{uk} = -2e_{ui}\, p_{uk}$$

因为我们现在有了梯度，我们可以为puk和qki应用更新规则：

$$p'_{uk} = p_{uk} - \alpha * \frac{\partial}{\partial puk}(e_{ui}^2) = p_{uk} + 2e_{ui}\, q_{ki}$$

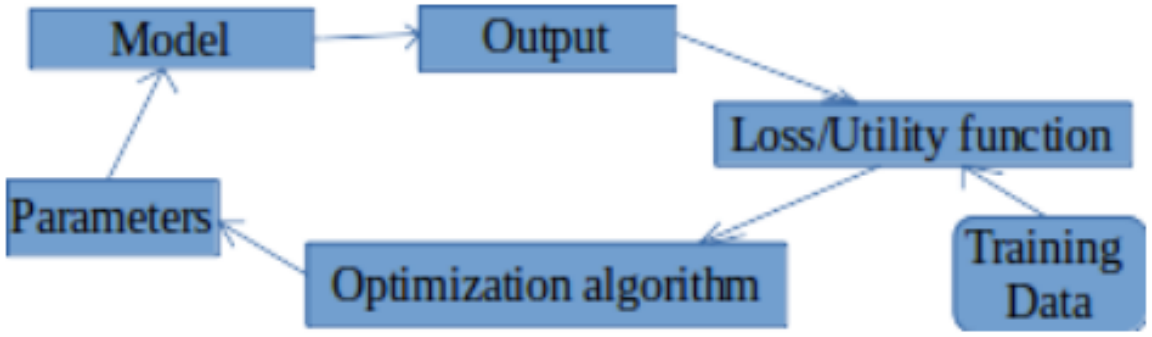
$$q'_{ki} = q_{ki} - \alpha * \frac{\partial}{\partial qki}(e_{ui}^2) = q_{ki} + 2e_{ui}\, p_{uk}$$

α是学习率，它决定每次更新的大小。重复上述更新过程，直到误差最小化为止，最后我们可以得到最优的P和Q矩阵，可以用来预测评分。我们快速回顾一下这个算法是如何工作的，然后我们来构建推荐引擎预测未评分电影的评分。

下面是矩阵分解预测评分的工作原理:

```
# for f = 1,2,....,k :
    # for rui ε R :
        # predict rui
        # update puk and qki
```

基于每个潜在特征，R矩阵中所有缺失的评分都将使用预测的rui值进行填充。然后利用梯度下降法对puk和qki进行更新，得到它们的最优值。过程如下图所示:



现在已经了解了这个算法的内部工作原理，接下来我们将举一个例子，看看如何将矩阵分解成它的组成部分。

使用一个2x3矩阵，如下图所示:

$$A = \begin{bmatrix} 3 & 2 & 2 \\ 2 & 3 & -2 \end{bmatrix}$$

这里我们有2个用户和3部电影的相应评分。现在，我们将这个矩阵分解成子部分，如下所示:

$$A_{2X3} = P_{2X2}\ \Sigma_{2X3}\ Q^{T}_{3X3}$$

AAT的特征值会给我们P矩阵而ATA的特征值会给我们Q矩阵，Σ是AAT或ATA矩阵特征值的平方根。

计算AAT的特征值:

$$AA^{T} = \begin{bmatrix} 17 & 8 \\ 8 & 17 \end{bmatrix}$$

$$\det(AA^T - \lambda I) = 0$$
$$\lambda^2 - 34\lambda + 225 = (\lambda - 25)(\lambda - 9) = 0$$

AAT的特征值是25，9同样道理，我们可以计算出ATA的特征值。这些值是25，9，0，现在我们可以计算AAT和ATA对应的特征向量。

特征值λ= 25,我们有:

$$A^TA - 25*I =$$

-12	12	2
12	-12	2
2	-2	-17

可以按行简化为:

1	-1	0
0	0	1
0	0	0

该矩阵内核中的单位向量为:

$$q_1 = \begin{bmatrix} 1/\sqrt{2} \\ 1/\sqrt{2} \\ 0 \end{bmatrix}$$

同样,λ= 9我们有:

$$A^TA - 9*I =$$

4	12	2
12	4	-2
2	-2	-1

可以按行简化为:

1	0	-1/4
0	1	1/4
0	0	0

该矩阵内核中的单位向量为:

$$q_2 = \begin{bmatrix} 2/3 \\ 2/3 \\ -1/3 \end{bmatrix}$$

对于最后一个特征向量，我们可以找到一个单位向量垂直于q1和q2。所以,

$$q_3 = \begin{bmatrix} 2/3 \\ 2/3 \\ -1/3 \end{bmatrix}$$

Σ2X3矩阵是AAT或ATA特征值的平方根即25和9:

$$\Sigma_{2X3} = \begin{bmatrix} 5 & 0 & 0 \\ 0 & 3 & 0 \end{bmatrix}$$

最后，我可可以通过公式σpi = Aqi, or pi = 1/σ(Aqi) 计算P2X2 :

$$P_{2X2} = \begin{bmatrix} 1/\sqrt{2} & 1/\sqrt{2} \\ 1/\sqrt{2} & 1/\sqrt{2} \end{bmatrix}$$

所以由A矩阵分解出的矩阵如下所示:

$$A = \begin{bmatrix} 1/\sqrt{2} & 1/\sqrt{2} \\ 1/\sqrt{2} & 1/\sqrt{2} \end{bmatrix} \begin{bmatrix} 5 & 0 & 0 \\ 0 & 3 & 0 \end{bmatrix} \begin{bmatrix} 1/\sqrt{2} & 1/\sqrt{2} & 0 \\ 1/\sqrt{18} & -1/\sqrt{18} & 4/\sqrt{18} \\ 2/3 & -2/3 & -1/3 \end{bmatrix}$$

我们有了P和Q矩阵，我们可以使用梯度下降法得到它们的优化版本，现在我们来使用矩阵分解来构建推荐引擎。

7 使用矩阵分解构建一个推荐引擎

我们首先定义一个函数来预测用户对所有未被他或她评分的所有电影的评分。

```
class MF():

    # Initializing the user-movie rating matrix, no. of latent features, alpha and beta.

    def __init__(self, R, K, alpha, beta, iterations):

        self.R = R

        self.num_users, self.num_items = R.shape

        self.K = K

        self.alpha = alpha

        self.beta = beta

        self.iterations = iterations

    # Initializing user-feature and movie-feature matrix

    def train(self):

        self.P = np.random.normal(scale=1./self.K, size=(self.num_users, self.K))

        self.Q = np.random.normal(scale=1./self.K, size=(self.num_items, self.K))        #
Initializing the bias terms

        self.b_u = np.zeros(self.num_users)

        self.b_i = np.zeros(self.num_items)

        self.b = np.mean(self.R[np.where(self.R != 0)])

        # List of training samples
```

```
        self.samples = [
            (i, j, self.R[i, j])
            for i in range(self.num_users)
            for j in range(self.num_items)
            if self.R[i, j] > 0
        ]
        # Stochastic gradient descent for given number of iterations
        training_process = []
        for i in range(self.iterations):
            np.random.shuffle(self.samples)
            self.sgd()
            mse = self.mse()
            training_process.append((i, mse))
            if (i+1) % 20 == 0:
                print("Iteration: %d ; error = %.4f" % (i+1, mse))
        return training_process

    # Computing total mean squared error
    def mse(self):
```

```
    xs, ys = self.R.nonzero()
    predicted = self.full_matrix()
    error = 0
    for x, y in zip(xs, ys):
      error += pow(self.R[x, y] - predicted[x, y], 2)
    return np.sqrt(error)


# Stochastic gradient descent to get optimized P and Q matrix
def sgd(self):
    for i, j, r in self.samples:
        prediction = self.get_rating(i, j)
        e = (r - prediction)
        self.b_u[i] += self.alpha * (e - self.beta * self.b_u[i])
        self.b_i[j] += self.alpha * (e - self.beta * self.b_i[j])
        self.P[i, :] += self.alpha * (e * self.Q[j, :] - self.beta * self.P[i,:])
        self.Q[j, :] += self.alpha * (e * self.P[i, :] - self.beta * self.Q[j,:])


# Ratings for user i and moive j
```

```
    def get_rating(self, i, j):
        prediction = self.b + self.b_u[i] + self.b_i[j] + self.P[i, :].dot(self.Q[j, :].T)
        return prediction

    # Full user-movie rating matrix

    def full_matrix(self):
        return mf.b + mf.b_u[:,np.newaxis] + mf.b_i[np.newaxis:,] + mf.P.dot(mf.Q.T)
```

现在我们有了一个可以预测评分的函数，这个函数的输入是：

- R-用户-电影评分矩阵
- K-潜在特征的个数
- Alpha-随机梯度下降的学习率
- Beta-正则化参数偏差
- Iterations-执行随机梯度下降的迭代次数

我们必须将用户-电影评分转换为矩阵形式，在python中使用pivot函数来完成转换。

```
R= np.array(ratings.pivot(index = 'user_id', columns ='movie_id', values = 'rating').fillna(0))
```

fillna(0)，表示将所有缺失值都用0来填充。现在我们有了R矩阵，可以初始化潜在特征的数量，但是这些特征的数量必须小于或等于原始特征的数量。

现在我们预测所有的缺失的评分

```
参数初始化为：K=20，alpha=0.001，

beta=0.01，迭代次数=100。

mf = MF(R, K=20, alpha=0.001, beta=0.01, iterations=100)

training_process = mf.train()

print()

print("P x Q:")

print(mf.full_matrix())

print()
```

下面给我们提供了每20次迭代后的误差值，最后是完整的用户-电影评分矩阵，输出是这样的：

```
Iteration: 20 ; error = 296.1205
Iteration: 40 ; error = 291.0587
Iteration: 60 ; error = 287.6521
Iteration: 80 ; error = 282.1805
Iteration: 100 ; error = 273.0630

P x Q:
[[3.8423072  3.22186837 3.14461749 ... 3.25195466 3.43969917 3.44625429]
 [3.98452024 3.33168034 3.14124963 ... 3.33857732 3.51054473 3.45930679]
 [3.21484895 2.71034092 2.52923913 ... 2.7549562  2.93259305 2.91370763]
 ...
 [4.24275975 3.64593082 3.41077192 ... 3.66791838 3.76867477 3.7662716 ]
 [4.39613375 3.8452877  3.57125971 ... 3.82585809 3.90684069 3.9031287 ]
 [3.6839097  3.26063154 3.16784624 ... 3.19645375 3.38092342 3.3155518 ]]
```

我们已经创建了我们的推荐引擎，接下来我们关注下一节如何评估推荐引擎的性能。

8 推荐引擎的评价指标

为了评估推荐引擎的性能，我们可以使用以下评价指标。

8.1 召回率

- 实际推荐的电影中，用户喜欢的比例有多少
- 公式如下：

$$\text{Recall} = \frac{tp}{tp + fn}$$

- 这里的tp表示推荐给用户电影中他或她喜欢的数量tp+fn表示他或她喜欢的总数量
- 如果一个用户喜欢5个电影而推荐引擎决定显示其中的3个，那么召回率就是0.6
- 召回率越大，推荐效果越好

8.2 精确度

- 在所有推荐的电影中，用户实际喜欢多少?
- 计算公式如下：

$$\text{Precision} = \frac{tp}{tp + fp}$$

- 这里的tp表示推荐给他/她喜欢的电影数量，tp+fp代表了向用户推荐的全部电影数量。
- 如果向用户推荐5部电影，他喜欢其中4部，那么精度将是0.8。

- 精度越高，推荐效果越好
- 但是考虑一下这样的情况：如果我们简单地推荐所有的电影，它们肯定会覆盖用户喜欢的电影。所以我们有百分之百的召回率！但是仔细考虑一下精确度，如果我们推荐1000部电影，而用户只喜欢其中的10部，那么精确度是0.1%，这真的很低。所以，我们的目标应该是最大化精确度和召回率。

8.3 均方误差（RMSE)

它衡量的是预测评分中的误差：

$$RMSE = \sqrt{\frac{\sum_{i=1}^{N} (Predicted_i - Actual_i)^2}{N}}$$

- Predicted 是模型预测的评分，Actual是原始评分
- 如果一个用户给一部电影打5分，我们预测它的打分是4，那么RMSE是1。
- RMSE越小，推荐效果越好

上面的指标告诉我们模型给出的推荐有多准确，但它们并不关注推荐的顺序，也就是说，它们不关注首先推荐的产品，以及之后的顺序。我们还需要一些度量标准，它们需要考虑了推荐产品的顺序。接下来我们来看看一些排名指标（ranking metrics）：

8.4 MRR（Mean Reciprocal Rank)

- **评估推荐的列表**

$$\mathrm{MRR} = \frac{1}{\mathrm{n}} \cdot \sum_{i=1}^{\mathrm{n}} \frac{1}{\mathrm{r}(\mathrm{Q}_i)}$$

- 假设我们已经向用户推荐了3部电影A，B，C，并且它们顺序是给定的。但是用户只喜欢电影C，因为电影C的等级是3，所以Reciprocal Rank是1/3。
- MRR越大，代表推荐效果越好

8.5 MAP at k（Mean Average Precision at cutoff k K位置截止的平均精度均值)

- 精确度和召回率并不关系推荐中的顺序
- 截止k的精度是通过只考虑从1到k推荐的子集来计算的精度

$$MAP_i = \frac{1}{|R_i|} \sum_{k=1}^{|R_i|} P(R_i[k])$$

- 假设我们已经给出了三个推荐，[0,1,1]。这里0表示推荐是不正确的，而1表示推荐是正确的。那么k的精度是[0 ,1/2, 2/3]，平均精度是（1/3）*(0+1/2+2/3）=0.38。
- 平均精度 越大，推荐的就越准确

8.6 NDCG（Normalized Discounted Cumulative Gain归一化累积折损增益

- MAP和NDCG的主要区别在于，MAP假设物品是感兴趣的（或者不是），而NDCG给出了相关性评分。
- 我们通过一个例子来理解它：假设在10部电影中——A到J，我们可以推荐前五部电影，即A、B、C、D和E，而我们不能推荐其他5部电影，也就是F，G，H，I和J，最终推荐是[A B C D]。所以在这个例子中NDCG将是1因为推荐的产品与用户相关。
- NDCG值越大，推荐效果越好

9 还可以尝试什么?

到目前为止，我们已经了解了什么是推荐引擎以及它的不同类型和它们的工作方式。基于内容的过滤和协同过滤算法都有各自的优点和缺点。

在某些领域，生成对商品的有用描述是非常困难的。如果用户之前的行为没有提供有用的信息，基于内容的推荐模型将不会选择该商品。我们还需要使用额外的技术，以便系统能够在用户已经显示出兴趣的范围之外给出推荐。

协同过滤模型没有这些缺点。因为不需要对所推荐的商品进行描述，系统可以处理任何类型的信息。此外，它还可以推荐用户以前没有兴趣的产品，但是，如果没有用户对新商品进行评分，那么协同过滤就不能为新商品进行推荐。即使用户开始对该商品进行评分，为了做出准确的推荐，也需要一段时间才能获得足够多的评分来做推荐。

一个将内容过滤和协同过滤结合起来的系统，可以潜在地从内容的表示和用户的相似性中获得更多信息。将基于内容的推荐和协同过滤的推荐进行加权平均，这是把协同性和基于内容过滤结合起来的一种方法。

这样做的各种方法有：

- 组合商品分数：

我们把从两种推荐方法中得到的评分组合起来，最简单的方式是取平均值。

假设有一种方法推荐对一部电影的评分为4，而另一种方法则推荐对同一部电影的评分为5。所以最终的建议是两个评级的平均值，也就是4 .5。

我们也可以给不同的方法分配不同的权重。

- 组合商品排名

假设协同过滤推荐了5部电影A、B、C、D和E，并且按以下顺序：A、B、C、D、E，而基于内容过滤则按照以下顺序推荐：B、D、A、C、E。

那么电影的顺序如下所示：

协同过滤：

Movie	Rank
A	1
B	0.8
C	0.6
D	0.4
E	0.2

基于内容过滤：

Movie	Rank
B	1
D	0.8
A	0.6
C	0.4
E	0.2

因此，一个混合推荐引擎将结合这些排名，并根据综合排名做出最终的推荐。合并后的排名是：

Movie	New Rank
A	1+0.6 = 1.6
B	0.8+1 = 1.8
C	0.6+0.4 = 1
D	0.4+0.8 = 1.2
E	0.2+0.2 = 0.4

最后的推荐也会基于这个排名，可以看到推荐的顺序会是：B, A, D, C, E。

通过这种方式，可以将两种或更多的方法组合起来，构建一个混合的推荐引擎，并提高它们的总体推荐精度和效率。

尾注

本文全面讲述了有关推荐引擎的内容，如果你想开始推荐引擎方面学习，本文也会是一个很好的帮助。我们不仅讨论了基本的推荐技术，而且还写到了如何实现当今业界的一些更先进的技术。

我们也针对每种技术联系到对应的现实问题，作为一个想要学习如何制作推荐引擎的人，我建议您学习本教程中讨论的技术，并在您的模型中实现它们。

你觉得这篇文章有用吗？可以在下方评论分享出你的观点！

（点击标题可跳转阅读）

用 Python 实现一个大数据搜索引擎

让我们一起来构建一个模板引擎（一）

让我们一起来构建一个模板引擎（二）

觉得本文对你有帮助？请分享给更多人

关注「Python开发者」加星标，提升Python技能

Python开发者

分享Python相关技术干货 · 资讯 · 高薪职位 · 教程

微信号：PythonCoder

长按识别二维码关注

伯乐在线 旗下微信公众号

商务合作QQ：2302462408