# 【機器學習】創建自己的電影推薦系統

機器學習初學者 今天

以下文章來源於磐創AI，作者Flin



作者| SO_HAM

編譯| Flin

來源| analyticsvidhya

## 介紹

“每次我去看電影，不管電影是關於什麼的，都很神奇。“——史蒂芬·斯皮爾伯格

每個人都喜歡電影，不分年齡、性別、種族、膚色或地理位置。通過這種神奇的媒介，我們在某種程度上彼此聯繫在一起。然而，最有趣的是，我們的選擇和組合在電影偏好方面是多麼獨特。

有些人喜歡特定類型的電影，比如驚悚片、愛情片或科幻片，而另一些人則喜歡主演和導演。當我們考慮到所有這些因素時，要概括一部電影並說每個人都會喜歡它是非常困難的。但儘管如此，我們仍然可以看到相似的電影受到社會特定人群的喜愛。

這就是我們作為數據科學家的作用，從觀眾的所有行為模式中提取核心信息，也從電影本身中提取信息。所以，廢話不多說，讓我們直接進入推薦系統的基礎。

## 什麼是推薦系統?

簡單地說，推薦系統是一個過濾程序，其主要目標是預測用戶對特定領域的項目或項目的“評級”或“偏好”。在我們的例子中，這個特定於領域的項目是一部電影，因此，我們推薦系統的主要重點是在給定用戶的一些數據的情況下，過濾和預測哪些是用戶更喜歡的電影。

## 有哪些不同的過濾策略?

![](http://qiniu.aihubs.net/88506recommendation system.png)

- **基於內容的過濾**

此过滤策略基于提供的关于项目的数据。该算法会推荐与用户过去喜欢的产品相似的产品。这种相似度(通常是余弦相似度)是根据我们拥有的关于商品的数据以及用户过去的偏好计算出来的。

例如，如果用户喜欢《The Prestige》这样的电影，那么我们就可以向他推荐克里斯蒂安·贝尔(Christian Bale)的电影、惊悚片(Thriller)或者克里斯托弗·诺兰(Christopher Nolan)导演的电影。

这里发生了什么？用户的推荐系统检查过去的喜好,找到这部电影《The Prestige》,然后试图找到类似的电影,使用数据库中的信息,如主演、导演、相关体裁的电影，制作公司等，基于这些信息找到类似于《The Prestige》的电影。

**缺点**

1. 用户很少能接触到不同类型的产品
2. 由于用户不尝试不同类型的产品，业务无法扩展。

- **协同过滤**

该过滤策略基于用户行为的组合，并将其与数据库中其他用户的行为进行比较和对比。所有用户的历史在该算法中扮演着重要的角色。基于内容的过滤和协同过滤的主要区别在于，协同过滤是所有用户与项目的交互影响推荐算法，而基于内容的过滤只考虑相关用户的数据。

协同过滤有多种实现方式，但需要把握的主要概念是，在协同过滤中，多个用户的数据会影响推荐的结果。而且建模并不仅仅依赖于一个用户的数据。

协同过滤算法有两种:

- **基于用户的协同过滤**

这里的基本理念是找到与用户“A”有相似偏好模式的用户，然后推荐那些“A”还没有遇到过的相似用户喜欢的商品。这是通过建立一个矩阵来实现的，矩阵中列出了每个用户根据其手头的任务对其进行评级/查看/喜欢/点击的项目，然后计算用户之间的相似度得分，最后推荐相关用户不知道但与他/她相似的用户喜欢的项目。

例如，如果用户A喜欢“Batman Begins”、“Justice League”和“the Avengers”，而用户B喜欢“Batman Begins”、“Justice League”和“Thor”，那么他们的兴趣是相似的，因为我们知道这些电影都属于超级英雄类型。因此，用户a很有可能会喜欢《雷神》，用户B很有可能会喜欢《复仇者联盟》。

**缺点**

1. 人是浮躁的，他们的喜好是不断变化的，因为这个算法是基于用户相似度的，它可能会挑选出两个用户之间最初的相似模式，一段时间后，可能会有完全不同的偏好。
2. 用户比项目多很多，因此维护这么大的矩阵变得非常困难，因此需要定期重新计算。

3. 该算法非常容易受到先令攻击，其中包含带有偏见的偏好模式的虚假用户档案被用来操纵关键决策。

- **基于项目协同过滤**

这种情况下的概念是找到相似的电影，而不是相似的用户，然后推荐与“A”过去喜欢的电影相似的电影。这是通过找到被同一用户评价/观看/点赞/点击的每一对物品，然后在所有同时评价/观看/点赞/点击的用户中测量那些被评价/观看/点赞/点击的物品的相似性，最后根据相似性分数推荐它们。

例如，我们选取两部电影“A”和“B”，并根据这两部电影的相似度，由所有给这两部电影都评级过的用户检查它们的评级，根据给这两部电影都评级过的用户的评级相似度，我们会发现相似的电影。所以，如果大多数普通用户对“A”和“B”的评价都是相似的，那么“A”和“B”很有可能是相似的，因此如果有人观看并喜欢“A”，那么他们就应该被推荐“B”，反之亦然。

## 优于基于用户的协同过滤

1. 不像人们的喜好千变万化，电影不会改变。

2. 矩阵的项通常比人少很多，因此更容易维护和计算矩阵。

3. 先令攻击更加困难，因为电影不能伪造。

**让我们开始编写我们自己的电影推荐系统**

在这个实现中，当用户搜索一部电影时，我们将使用我们的电影推荐系统推荐排名前10的类似电影。我们将使用基于项目的协同过滤算法。本演示中使用的数据集是movielens-small数据集。

- movielens-small数据集：https://www.kaggle.com/shubhammehta21/movie-lens-small-latest-dataset

## 启动并运行数据

首先，我们需要导入我们将在电影推荐系统中使用的库。另外，我们将通过添加CSV文件的路径来导入数据集。

```
import pandas as pd
import numpy as np
from scipy.sparse import csr_matrix
from sklearn.neighbors import NearestNeighbors
import matplotlib.pyplot as plt
import seaborn as sns
movies = pd.read_csv("../input/movie-lens-small-latest-dataset/movies.csv")
ratings = pd.read_csv("../input/movie-lens-small-latest-dataset/ratings.csv")
```

现在我们已经添加了数据，让我们看看这些文件，使用dataframe.head()命令打印数据集的前5行。

让我们来看看电影数据集:

```
movies.head()
```

| | movieId | title | genres |
|---|---|---|---|
| 0 | 1 | Toy Story (1995) | Adventure\|Animation\|Children\|Comedy\|Fantasy |
| 1 | 2 | Jumanji (1995) | Adventure\|Children\|Fantasy |
| 2 | 3 | Grumpier Old Men (1995) | Comedy\|Romance |
| 3 | 4 | Waiting to Exhale (1995) | Comedy\|Drama\|Romance |
| 4 | 5 | Father of the Bride Part II (1995) | Comedy |

电影数据集有:

- movieId——推荐完成后，我们会得到一个包含所有相似movieId的列表，并从这个数据集获得每个电影的标题。
- genres，体裁——这个过滤方法不需要。

```
ratings.head ()
```

| | userId | movieId | rating | timestamp |
|---|---|---|---|---|
| 0 | 1 | 1 | 4.0 | 964982703 |
| 1 | 1 | 3 | 4.0 | 964981247 |
| 2 | 1 | 6 | 4.0 | 964982224 |
| 3 | 1 | 47 | 5.0 | 964983815 |
| 4 | 1 | 50 | 5.0 | 964982931 |

评级数据集具有：

- userId——对每个用户都是唯一的。
- movieId——使用这个特性，我们从电影数据集获取电影的标题。
- rating——每个用户给所有电影的评级，使用这个我们将预测前10个类似的电影。

在这里，我们可以看到userId 1观看了movieId 1和3，并将它们都评为4.0，但根本没有给movieId 2打分。这个解释很难从这个数据帧中提取出来。

因此，为了使事情更容易理解和使用，我们将创建一个新的数据帧，其中每个列将表示每个惟一的用户id，每个行表示每个惟一的movieId。

```
final_dataset = ratings.pivot(index='movieId',columns='userId',values='rating')
final_dataset.head()
```

| userId | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | ... | 601 | 602 | 603 | 604 | 605 | 606 | 607 | 608 | 609 | 610 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| movieId | | | | | | | | | | | | | | | | | | | | | |
| 1 | 4.0 | NaN | NaN | NaN | 4.0 | NaN | 4.5 | NaN | NaN | NaN | ... | 4.0 | NaN | 4.0 | 3.0 | 4.0 | 2.5 | 4.0 | 2.5 | 3.0 | 5.0 |
| 2 | NaN | NaN | NaN | NaN | NaN | 4.0 | NaN | 4.0 | NaN | NaN | ... | NaN | 4.0 | NaN | 5.0 | 3.5 | NaN | NaN | 2.0 | NaN | NaN |
| 3 | 4.0 | NaN | NaN | NaN | NaN | 5.0 | NaN | NaN | NaN | NaN | ... | NaN | NaN | NaN | NaN | NaN | NaN | NaN | 2.0 | NaN | NaN |
| 4 | NaN | NaN | NaN | NaN | NaN | 3.0 | NaN | NaN | NaN | NaN | ... | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN |
| 5 | NaN | NaN | NaN | NaN | NaN | 5.0 | NaN | NaN | NaN | NaN | ... | NaN | NaN | NaN | 3.0 | NaN | NaN | NaN | NaN | NaN | NaN |

5 rows × 610 columns

现在，更容易理解的是，userId 1对movieId 1和3进行了评级，但根本没有对movieId 3、4、5进行评级(因此它们被表示为NaN)，因此它们的评级数据是缺失的。

让我们解决这个问题，并将NaN归为0，以使算法更容易理解，同时也使数据看起来更令人舒服。

```
final_dataset.fillna(0,inplace=True)
final_dataset.head()
```

| userId | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | ... | 601 | 602 | 603 | 604 | 605 | 606 | 607 | 608 | 609 | 610 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| movieId | | | | | | | | | | | | | | | | | | | | | |
| 1 | 4.0 | 0.0 | 0.0 | 0.0 | 4.0 | 0.0 | 4.5 | 0.0 | 0.0 | 0.0 | ... | 4.0 | 0.0 | 4.0 | 3.0 | 4.0 | 2.5 | 4.0 | 2.5 | 3.0 | 5.0 |
| 2 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 4.0 | 0.0 | 4.0 | 0.0 | 0.0 | ... | 0.0 | 4.0 | 0.0 | 5.0 | 3.5 | 0.0 | 0.0 | 2.0 | 0.0 | 0.0 |
| 3 | 4.0 | 0.0 | 0.0 | 0.0 | 0.0 | 5.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 2.0 | 0.0 | 0.0 |
| 4 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 3.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| 5 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 5.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 | 0.0 | 3.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |

5 rows × 610 columns

## 去除数据中的噪音

在现实世界中，评分非常少，数据点大多来自非常受欢迎的电影和高参与度的用户。我们不希望电影被一小部分用户评分，因为它不够可信。同样，只给少数几部电影打分的用户也不应该被考虑在内。

因此，考虑到所有这些因素和一些反复试验，我们将通过为最终数据集添加一些过滤器来减少噪声。

- 至少有10个用户对一部电影进行了投票。
- 为了使一个用户有资格，至少50部电影应该由用户投票。

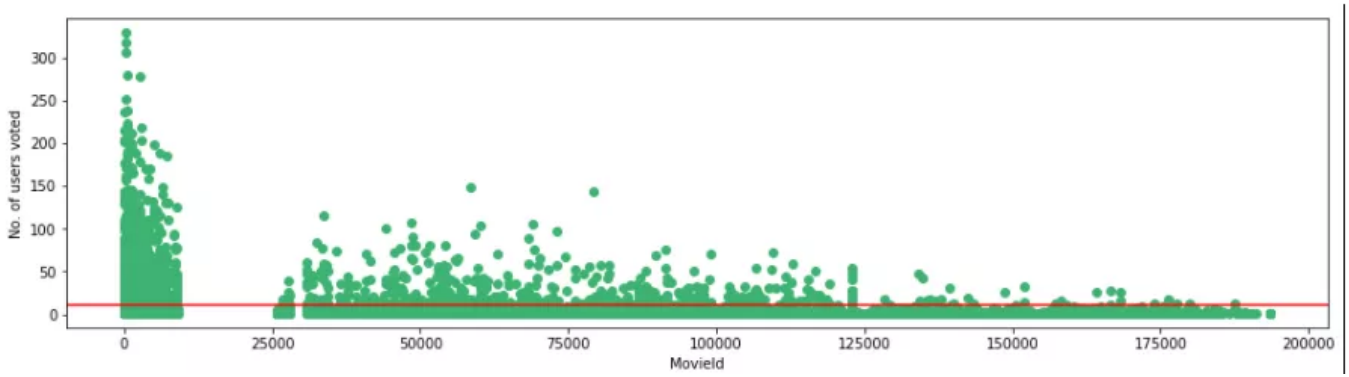
## 让我们直观地看到这些过滤器的外观

汇总投票的用户数量和被投票的电影数量。

```
no_user_voted = ratings.groupby('movieId')['rating'].agg('count')
no_movies_voted = ratings.groupby('userId')['rating'].agg('count')
```

让我们直观地看到以阈值10投票的用户数量。

```
f,ax = plt.subplots(1,1,figsize=(16,4))
# ratings['rating'].plot(kind='hist')
plt.scatter(no_user_voted.index,no_user_voted,color='mediumseagreen')
plt.axhline(y=10,color='r')
plt.xlabel('MovieId')
plt.ylabel('No. of users voted')
plt.show()
```

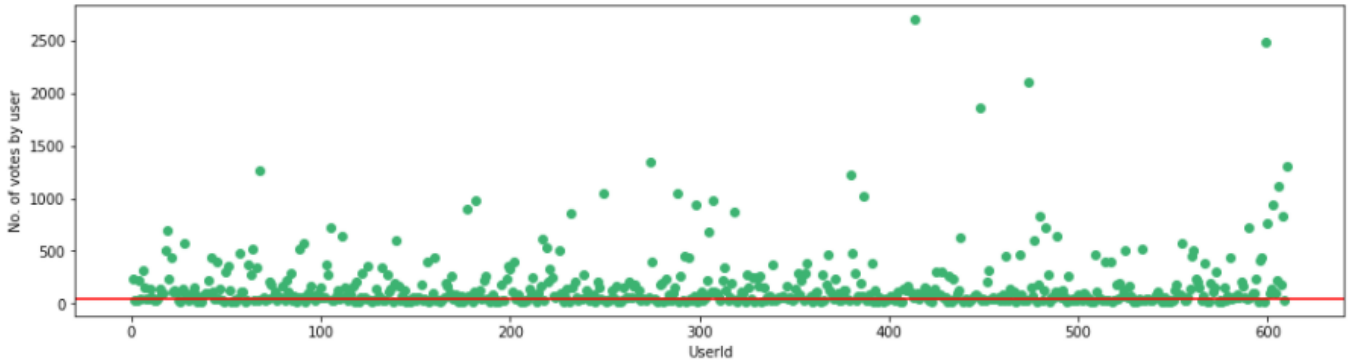



根据阈值设置进行必要的修改。

```
final_dataset = final_dataset.loc[no_user_voted[no_user_voted > 10].index,:]
```

让我们以50的阈值来可视化每个用户的投票数量。

```
f,ax = plt.subplots(1,1,figsize=(16,4))
plt.scatter(no_movies_voted.index,no_movies_voted,color='mediumseagreen')
plt.axhline(y=50,color='r')
plt.xlabel('UserId')
plt.ylabel('No. of votes by user')
plt.show()
```




根据阈值设置进行必要的修改。

```
final_dataset=final_dataset.loc[:,no_movies_voted[no_movies_voted > 50].index]
final_dataset
```

| userId | 1 | 4 | 6 | 7 | 10 | 11 | 15 | 16 | 17 | 18 | ... | 600 | 601 | 602 | 603 | 604 | 605 | 606 | 607 | 608 | 610 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| movieId | | | | | | | | | | | | | | | | | | | | | |
| 1 | 4.0 | 0.0 | 0.0 | 4.5 | 0.0 | 0.0 | 2.5 | 0.0 | 4.5 | 3.5 | ... | 2.5 | 4.0 | 0.0 | 4.0 | 3.0 | 4.0 | 2.5 | 4.0 | 2.5 | 5.0 |
| 2 | 0.0 | 0.0 | 4.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 3.0 | ... | 4.0 | 0.0 | 4.0 | 0.0 | 5.0 | 3.5 | 0.0 | 0.0 | 2.0 | 0.0 |
| 3 | 4.0 | 0.0 | 5.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 2.0 | 0.0 |
| 5 | 0.0 | 0.0 | 5.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 2.5 | 0.0 | 0.0 | 0.0 | 3.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| 6 | 4.0 | 0.0 | 4.0 | 0.0 | 0.0 | 5.0 | 0.0 | 0.0 | 0.0 | 4.0 | ... | 0.0 | 0.0 | 3.0 | 4.0 | 3.0 | 0.0 | 0.0 | 0.0 | 0.0 | 5.0 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 174055 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 0.0 | 4.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| 176371 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 0.0 | 4.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| 177765 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 0.0 | 4.5 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| 179819 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| 187593 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |

2121 rows × 378 columns

## 消除稀疏

我们的final_dataset的维数是2121 * 378，其中大多数值是稀疏的。我们只使用了一个小的数据集，但是对于电影镜头的原始大数据集，有超过100000个特征，我们的系统可能会在将这些特征输入到模型时耗尽计算资源。为了减少稀疏性，我们使用scipy库中的csr_matrix函数。

我将举个例子来说明它是如何工作的:

```
sample = np.array([[0,0,3,0,0],[4,0,0,0,2],[0,0,0,0,1]])
sparsity = 1.0 - ( np.count_nonzero(sample) / float(sample.size) )
print(sparsity)
```

```
0.7333333333333334
```

```
sample = np.array([[0,0,3,0,0],[4,0,0,0,2],[0,0,0,0,1]])
sparsity = 1.0 - ( np.count_nonzero(sample) / float(sample.size) )
print(sparsity)
```

```
  (0, 2)        3
  (1, 0)        4
  (1, 4)        2
  (2, 4)        1
```

正如你所看到的，csr_sample中没有稀疏值，值被分配为行和列索引。对于第0行和第2列，值是3。

应用csr_matrix函数到数据集:

```
csr_data = csr_matrix(final_dataset.values)
final_dataset.reset_index(inplace=True)
```

## 制作电影推荐系统模型

我们将使用KNN算法计算与余弦距离度量的相似度，这是非常快的，比皮尔逊系数更好。

```
knn = NearestNeighbors(metric='cosine', algorithm='brute', n_neighbors=20, n_jobs=-1)
```

```
knn.fit(csr_data)
```

## 推荐函数的制作

工作原理很简单。我们首先检查输入的电影名是否在数据库中，如果在数据库中，我们使用推荐系统查找相似的电影，并根据它们的相似距离对它们进行排序，然后只输出与输入电影之间的距离最高的10部电影

```
def get_movie_recommendation(movie_name):
   n_movies_to_reccomend = 10
   movie_list = movies[movies['title'].str.contains(movie_name)]
   if len(movie_list):
      movie_idx= movie_list.iloc[0]['movieId']
      movie_idx = final_dataset[final_dataset['movieId'] == movie_idx].index[0]
      distances , indices = knn.kneighbors(csr_data[movie_idx],n_neighbors=n_movies_to_reccomend+1)
      rec_movie_indices = sorted(list(zip(indices.squeeze().tolist(),distances.squeeze().tolist())),key=lambda x: x[1])[:0:-1]
      recommend_frame = []
      for val in rec_movie_indices:
         movie_idx = final_dataset.iloc[val[0]]['movieId']
         idx = movies[movies['movieId'] == movie_idx].index
         recommend_frame.append({'Title':movies.iloc[idx]['title'].values[0],'Distance':val[1]})
      df = pd.DataFrame(recommend_frame,index=range(1,n_movies_to_reccomend+1))
      return df
   else:
      return "No movies found. Please check your input"
```

**最后，我们来推荐一些电影吧!**

```
get_movie_recommendation('Iron Man')
```

我个人认为结果相当不错。所有在顶端的电影都是超级英雄或动画电影，就像输入电影“钢铁侠”一样，是孩子们的理想选择。

| | Title | Distance |
|---|---|---|
| 1 | Up (2009) | 0.368857 |
| 2 | Guardians of the Galaxy (2014) | 0.368758 |
| 3 | Watchmen (2009) | 0.368558 |
| 4 | Star Trek (2009) | 0.366029 |
| 5 | Batman Begins (2005) | 0.362759 |
| 6 | Avatar (2009) | 0.310893 |
| 7 | Iron Man 2 (2010) | 0.307492 |
| 8 | WALL·E (2008) | 0.298138 |
| 9 | Dark Knight, The (2008) | 0.285835 |
| 10 | Avengers, The (2012) | 0.285319 |

让我们再试一个:

```
get_movie_recommendation('Memento')
```

| | Title | Distance |
|---|---|---|
| 1 | American Beauty (1999) | 0.389346 |
| 2 | American History X (1998) | 0.388615 |
| 3 | Pulp Fiction (1994) | 0.386235 |
| 4 | Lord of the Rings: The Return of the King, The... | 0.371622 |
| 5 | Kill Bill: Vol. 1 (2003) | 0.350167 |
| 6 | Lord of the Rings: The Two Towers, The (2002) | 0.348358 |
| 7 | Eternal Sunshine of the Spotless Mind (2004) | 0.346196 |
| 8 | Matrix, The (1999) | 0.326215 |
| 9 | Lord of the Rings: The Fellowship of the Ring,... | 0.316777 |
| 10 | Fight Club (1999) | 0.272380 |

排名前十的电影都是严肃的、用心的电影，就像《记忆碎片》本身一样，所以我认为这个结果也是好的。

我们的模型运行得很好——一个基于用户行为的电影推荐系统。因此，我们在此总结我们的协同过滤。你可以在这里得到完整的实现代码。

- https://github.com/So-ham/Movie-Recommendation-System

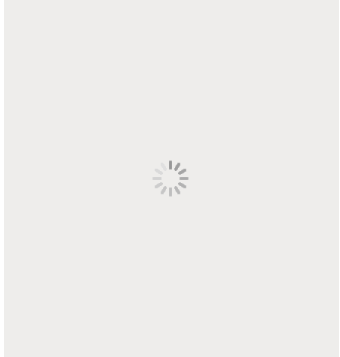
原文链接：https://www.analyticsvidhya.com/blog/2020/create-your-own-movie-movie-recommendation-system/


往期精彩回顾

- 适合初学者入门人工智能的路线及资料下载

- 机器学习及深度学习笔记等资料打印
- 机器学习在线手册
- 深度学习笔记专辑
- 《统计学习方法》的代码复现专辑
- AI基础下载
- 机器学习的数学基础专辑
- 温州大学《机器学习课程》视频

本站qq群851320808，加入微信群请扫码：



喜欢此内容的人还喜欢

「Python數據分析系列」1. 數據科學基本介紹

數據與智能


將偷懶進行到底，實現雙擊直接打開.ipynb文件

凹凸數據

2021年計算機視覺上採助學習路線

深度學習與計算機視覺