Getting Started with a Movie Recommendation System ｜ 电影推荐系统的入门

Brief ｜ 介绍

```
项目名称：电影推荐入门 （Getting Started with a Movie Recommendation System）
项目数据集来源 ： kaggle
项目原地址： https://www.kaggle.com/ibtesama/getting-started-with-a-movie-recommendation-system
成员列表 ： 华邵钶 （共一名）
目标： 实现电影推荐，比较类似于过滤器?
采用的算法技术： TF-IDF,余弦相似度，数据清洗的方法，协同过滤
```

Start of Movie Recommender System ｜开始学习电影推荐系统

```
The rapid growth of data collection has led to a new era of information. Data is being used to create more efficient systems and this is where Recommendation Systems come into play. Recommendation Systems are a type of information filtering systems as they improve the quality of search results and provides items that are more relevant to the search item or are realted to the search history of the user.
```

```
数据收集的快速增长导致了一个新的信息时代的到来。数据正被用来创建更有效的系统，这就是推荐系统发挥作用的地方。推荐系统是一种信息过滤系统，因为它们提高了搜索结果的质量，并提供与搜索项目更相关的项目或与用户的搜索历史相关的项目。
```

```
They are used to predict the rating or preference that a user would give to an item. Almost every major tech company has applied them in some form or the other: Amazon uses it to suggest products to customers, YouTube uses it to decide which video to play next on autoplay, and Facebook uses it to recommend pages to like and people to follow. Moreover, companies like Netflix and Spotify depend highly on the effectiveness of their recommendation engines for their business and sucees.
```

```
它们被用来预测用户对一个项目的评价或偏好。几乎每个主要的科技公司都以某种形式应用它们。亚马逊用它来向客户推荐产品，YouTube用它来决定下一个自动播放的视频，而Facebook用它来推荐喜欢的网页和关注的人。此外，像Netflix和Spotify这样的公司高度依赖其推荐引擎对其业务和服务的有效性。
```

KeyPoint of Start of Recommender System | 关于开始的关键点

- **rating** 评分
- **preference** 偏好

Tasks Introduction | 任务介绍

```
In this kernel we'll be building a baseline Movie Recommendation System using TMDB 5000 Movie Dataset. For novices like me this kernel will pretty much serve as a foundation in recommendation systems and will provide you with something to start with.
```

```
在这个内核中，我们将使用TMDB 5000电影数据集建立一个基线电影推荐系统。对于像我这样的新手来说，这个内核几乎可以作为推荐系统的基础，并为你提供一些开始。
```

Category of Recommender System | 分类系统介绍

Demographic Filtering | （人口统计学过滤/热度推荐）

```
Demographic Filtering- They offer generalized recommendations to every user, based on movie popularity and/or genre. The System recommends the same movies to users with similar demographic features. Since each user is different , this approach is considered to be too simple. The basic idea behind this system is that movies that are more popular and critically acclaimed will have a higher probability of being liked by the average audience.
```

```
人口统计学过滤--他们根据电影的受欢迎程度和/或类型，向每个用户提供概括性的推荐。该系统向具有类似人口统计特征的用户推荐相同的电影。由于每个用户都是不同的，这种方法被认为是过于简单。这个系统背后的基本想法是，更受欢迎和受到好评的电影将有更高的概率被普通观众所喜欢。
```

Keypoint of Demographic Filtering | 人工统计学过滤的重点

- 基于电影的热门程度进行推荐，类似于热度榜单，热搜
- 问题在于每个人都是不同的，这种算法被认为过于简单

Content Based Filtering | （基于内容的过滤/相同类型推荐）

```
They suggest similar items based on a particular item. This system uses item metadata, such as genre, director, description, actors, etc. for movies, to make these recommendations. The general idea behind these recommender systems is that if a person liked a particular item, he or she will also like an item that is similar to it.
```

> 基于内容的过滤--他们根据特定的项目来推荐类似的项目。这个系统使用项目元数据，如电影的类型、导演、描述、演员等，来进行这些推荐。这些推荐系统背后的一般想法是，如果一个人喜欢某项物品，他或她也会喜欢与之相似的物品。

Keypoint of Content Based Filtering | 基于内容过滤的重点

- 人类会喜欢相同的类型的事物，类似于视频下方的推荐

Collaborative Filtering | （协作过滤 / 推荐你相同兴趣人看的东西）

> This system matches persons with similar interests and provides recommendations based on this matching. Collaborative filters do not require item metadata like its content-based counterparts.

> 协作过滤--该系统将具有类似兴趣的人进行匹配，并根据这种匹配提供推荐。协作式过滤器不需要像其基于内容的同行那样的项目元数据。

Keypoint of Collaborative Filtering | 基于协作过滤的重点

- 将与你兴趣相同人看的内容推奖给你
- 不需要源数据就能实现推荐

Code Preparation | 准备工作

Import Package and Dataset | 导入库和数据库

```
import pandas as pd
import numpy as np
# 请注意这边的路径为绝对路径
df1 = pd.read_csv('/Users/llonvne/OneDrive/大数据挖掘/大作业/电影推荐入门/dataset/dataset/tmdb_dataset/tmdb_5000_credits.csv')
df2 = pd.read_csv('/Users/llonvne/OneDrive/大数据挖掘/大作业/电影推荐入门/dataset/dataset/tmdb_dataset/tmdb_5000_movies.csv')
```

Dataset introduction | 数据集合介绍

Dataset : tmdb_5000_credits.csv | TMDB电影数据集

```
The first dataset contains the following features:-

movie_id - A unique identifier for each movie.
cast - The name of lead and supporting actors.
crew - The name of Director, Editor, Composer, Writer etc.
```

```
第一个数据集包含以下特征：-

movie_id - 每部电影的唯一标识符。
cast - 主演和配角的名字。
剧组--导演、编辑、作曲家、编剧等的名字。
```

Keypoint of Dataset : tmdb_5000_credits.csv | TMDB电影数据集的重点

- movie_id 标识符号
- cast 主演与配角
- crew 导演编剧等

Dataset : tmdb_5000_movies.csv

```
The second dataset has the following features:-

budget - The budget in which the movie was made.
genre - The genre of the movie, Action, Comedy ,Thriller etc.
homepage - A link to the homepage of the movie.
id - This is infact the movie_id as in the first dataset.
keywords - The keywords or tags related to the movie.
original_language - The language in which the movie was made.
original_title - The title of the movie before translation or adaptation.
overview - A brief description of the movie.
popularity - A numeric quantity specifying the movie popularity.
production_companies - The production house of the movie.
production_countries - The country in which it was produced.
release_date - The date on which it was released.
revenue - The worldwide revenue generated by the movie.
runtime - The running time of the movie in minutes.
status - "Released" or "Rumored".
tagline - Movie's tagline.
title - Title of the movie.
vote_average - average ratings the movie recieved.
vote_count - the count of votes recieved.
```

```
第二个数据集有以下特征：-

预算 - 电影制作的预算。
类型 - 电影的类型，动作片、喜剧片、惊悚片等。
homepage - 电影主页的链接。
id - 这实际上是第一个数据集中的movie_id。
keywords - 与电影有关的关键词或标签。
original_language - 电影所使用的语言。
original_title - 翻译或改编前的电影名称。
overview - 电影的简要描述。
```

```
popularity – 一个指定电影受欢迎程度的数字量。
production_companies – 电影的制作公司。
production_countries – 制作电影的国家。
release_date – 它被释放的日期。
revenue – 电影在全世界产生的收入。
runtime – 电影的运行时间（分钟）。
status – "Released" or "Rumored"。
tagline – 电影的标语。
title – 电影的标题。
vote_average – 电影获得的平均评分。
vote_count – 收到的投票数。
```

Linking data collections with IDs 用 ID 链接两个数据集合

```
df1.columns = ['id','tittle','cast','crew']
df2= df2.merge(df1,on='id')
```

```
将 df1 的数据链接到 df2 用 ID链接
```

keypoint of Linking data collections with IDS | 关于表合并的重点

- 表合并 merge 函数通过某一个关键指标合并两个表

Demographic Filtering - 人工统计学过滤

Preparation for Demographic Filtering | 准备工作

```
Before getting started with this -

we need a metric to score or rate movie
Calculate the score for every movie
Sort the scores and recommend the best rated movie to the users.
```

```
在开始研究这个问题之前 -

我们需要一个衡量标准来给电影打分或评级
计算每部电影的分数
对分数进行排序，并向用户推荐最佳评分的电影。
```

Keypoint of Preparation for Demographic Filtering | 人工统计学准备工作的重点

- 需要一个衡量标准给电影评分
- 计算分数
- 排序

Evaluation function of Movie | 电影评价函数

$$\text{Weighted Rating (WR)} = \left(\frac{v}{v+m}.R\right) + \left(\frac{m}{v+m}.C\right)$$

Introduction of Evaluation function of Movie ｜ 电影评价函数简介

```
where,

v is the number of votes for the movie;
m is the minimum votes required to be listed in the chart;
R is the average rating of the movie; And
C is the mean vote across the whole report
```

```
其中：
v是电影的票数。
m是在图表中列出的最低票数要求。
R是电影的平均评分；以及
C是整个报告的平均票数
```

Keypoint of Evaluation function of Movies | 电影评价函数的重点

- 电影的票数
- 最低票数的要求
- 电影的平均评分
- 整个报告平均票数

Calculate C which means the mean vote across the whole report ｜ 计算报告的平均评分

Code for Calculate C ｜ 计算平均票数的代码

```
C= df2['vote_average'].mean()
print("Vote Average is ",C)
```

Result of Code for Calculate C ｜代码的结果

```
Vote Average is  6.092171559442016
```

Keypoint of Code for Calculate C ｜ 关键点

- mean() 方法

Determine m based on the average score ｜ 根据平均分数来确定最低票数的要求

How to determine m | 如何确定最低票数要求

```
So, the mean rating for all the movies is approx 6 on a scale of 10.The next step is to determine an appropriate value for m, the minimum votes required to be listed in the chart. We will use 90th percentile as our cutoff. In other words, for a movie to feature in the charts, it must have more votes than at least 90% of the movies in the list.
```

```
因此，所有电影的平均评分在10分制中约为6分。下一步是确定一个适当的m值，即在图表中列出所需的最低票数。我们将使用第90个百分点作为我们的分界线。换句话说，一部电影要想出现在排行榜上，它的票数至少要比排行榜上90%的电影多。
```

Code for Calculate m ｜ 计算 m 的代码

```
m= df2['vote_count'].quantile(0.9)
print("the minimum votes requires is",m)
```

Result of Code for Calculate m | 代码的结果

```
the minimum votes requires is 1838.4000000000015
```

Keypoint of Code for Calculate m | 计算 m 关键点

- quantile() 方法

Screening out movies that meet the requirements ｜ 筛选出满足要求的数据

Code for Filtering | 筛选代码实现

```
q_movies = df2.copy().loc[df2['vote_count'] >= m]
q_movies.shape
```

Result of Code for Filtering | 筛选的结果展示

Commandline Output | 命令行输出

```
(481, 23) # 第一个为行数，第一位列数
```

Variable display | 变量展示

Movie_Recommender.ipynb > q_movies (481, 23)

	index	budget	genres	homepa…	id	keywor…	origin…	origin…	overvi…	popula…	produc…
0	0	237000000	[{"id": …	http://w…	19995	[{"id": …	en	Avatar	In the 2…	150.4375…	[{"name"
1	1	300000000	[{"id": …	http://d…	285	[{"id": …	en	Pirates …	Captain …	139.0826…	[{"name"
2	2	245000000	[{"id": …	http://w…	206647	[{"id": …	en	Spectre	A crypti…	107.3767…	[{"name"
3	3	250000000	[{"id": …	http://w…	49026	[{"id": …	en	The Dark…	Followin…	112.31295	[{"name"
4	4	260000000	[{"id": …	http://m…	49529	[{"id": …	en	John Car…	John Car…	43.926995	[{"name"
5	5	258000000	[{"id": …	http://w…	559	[{"id": …	en	Spider-M…	The seem…	115.6998…	[{"name"
6	6	260000000	[{"id": …	http://d…	38757	[{"id": …	en	Tangled	When the…	48.681969	[{"name"
7	7	280000000	[{"id": …	http://m…	99861	[{"id": …	en	Avengers…	When Ton…	134.2792…	[{"name"
8	8	250000000	[{"id": …	http://h…	767	[{"id": …	en	Harry Po…	As Harry…	98.885637	[{"name"
9	9	250000000	[{"id": …	http://w…	209112	[{"id": …	en	Batman v…	Fearing …	155.7904…	[{"name"
10	11	200000000	[{"id": …	http://w…	10764	[{"id": …	en	Quantum …	Quantum …	107.9288…	[{"name"
11	12	200000000	[{"id": …	http://d…	58	[{"id": …	en	Pirates …	Captain …	145.8473…	[{"name"
12	13	255000000	[{"id":	http://d…	57201	[{"id":	en	The Lone	The Texa…	49.046956	[{"name"

Keypoint of Code for Filtering | 筛选数据的关键

- 判断票数是否大于 m

Apply Evaluation function to Qualified films | 在满足要求的电影上应用评价函数

Start for Constructors Function | 构造评价函数

```
We see that there are 481 movies which qualify to be in this list. Now, we need to calculate our metric for each qualified movie. To do this, we will define a function, weighted_rating() and define a new feature score, of which we'll calculate the value by applying this function to our DataFrame of qualified movies:
```

```
我们看到，有481部电影有资格进入这个名单。现在，我们需要为每部合格的电影计算我们的指标。要做到这一点，我们将定义一个函数，weighted_rating()，并定义一个新的特征分数，我们将通过对合格电影的DataFrame应用这个函数来计算其数值。
```

Code for Evaluation Function | 评价函数代码实现

Define Evaluation Score Calculate Function | 定义并且实现评价函数

```
def weighted_rating(x, m=m, C=C):
    v = x['vote_count']
    R = x['vote_average']
    # Calculation based on the IMDB formula
    return (v/(v+m) * R) + (m/(m+v) * C)
```

Calculate scores for all movies and add them to the score column | 为每一个电影计算评价函数

```
# Define a new feature 'score' and calculate its value with `weighted_rating()`
q_movies['score'] = q_movies.apply(weighted_rating, axis=1)
```

Result for Code for Evaluation () | 输出评价函数结果

Movie_Recommender.ipynb > q_movies (481, 24)

venue	runtime	spoken…	status	tagline	title	vote_a…	vote_c…	tittle	cast	crew	score
341469	142	[{"iso_6…	Released	Fear can…	The Shaw…	8.5	8205	The Shaw…	[{"cast_…	[{"credi…	8.059257…
0853753	139	[{"iso_6…	Released	Mischief…	Fight Cl…	8.3	9413	Fight Cl…	[{"cast_…	[{"credi…	7.939256…
045584…	152	[{"iso_6…	Released	Why So S…	The Dark…	8.2	12002	The Dark…	[{"cast_…	[{"credi…	7.920020…
3928762	154	[{"iso_6…	Released	Just bec…	Pulp Fic…	8.3	8428	Pulp Fic…	[{"cast_…	[{"credi…	7.904645…
5532764	148	[{"iso_6…	Released	Your min…	Inception	8.1	13752	Inception	[{"cast_…	[{"credi…	7.863239…
5066411	175	[{"iso_6…	Released	An offer…	The Godf…	8.4	5893	The Godf…	[{"cast_…	[{"credi…	7.851236…
5120017	169	[{"iso_6…	Released	Mankind …	Interste…	8.1	10867	Interste…	[{"cast_…	[{"credi…	7.809478…
7945399	142	[{"iso_6…	Released	The worl…	Forrest …	8.2	7927	Forrest …	[{"cast_…	[{"credi…	7.803187…
188889…	201	[{"iso_6…	Released	The eye …	The Lord…	8.1	8064	The Lord…	[{"cast_…	[{"credi…	7.727242…
8400000	124	[{"iso_6…	Released	The Adve…	The Empi…	8.2	5879	The Empi…	[{"cast_…	[{"credi…	7.697883…
1368364	178	[{"iso_6…	Released	One ring…	The Lord…	8	8705	The Lord…	[{"cast_…	[{"credi…	7.667341…
5398007	121	[{"iso_6…	Released	A long t…	Star Wars	8.1	6624	Star Wars	[{"cast_…	[{"credi…	7.663812…
1365567	195	[{"iso_6	Released	Whoever	Schindle	8.3	4329	Schindle	[{"cast	[{"credi	7.641882

```
Notice: The sorting is done manually by VScode, this data itself is not sorted!
```

Print Finally Result | 输出结果

```
Finally, let's sort the DataFrame based on the score feature and output the title,
vote count, vote average and weighted rating or score of the top 10 movies.
```

```
最后，让我们根据分数特征对DataFrame进行排序，并输出前10部电影的标题、投票数、平均票数和加权评级或分数。
```

Code for Print Result | 输出结果的代码

```
# Sort movies based on score calculated above | 按照 score 排序
q_movies = q_movies.sort_values('score', ascending=False)

# Print the top 15 movies | 打印
q_movies[['title', 'vote_count', 'vote_average', 'score']].head(10)
```

Result for Code for Print Result | 输出评分的结果的截图

	title	vote_count	vote_average	score
1881	The Shawshank Redemption	8205	8.5	8.059258
662	Fight Club	9413	8.3	7.939256
65	The Dark Knight	12002	8.2	7.920020
3232	Pulp Fiction	8428	8.3	7.904645
96	Inception	13752	8.1	7.863239
3337	The Godfather	5893	8.4	7.851236
95	Interstellar	10867	8.1	7.809479
809	Forrest Gump	7927	8.2	7.803188
329	The Lord of the Rings: The Return of the King	8064	8.1	7.727243
1990	The Empire Strikes Back	5879	8.2	7.697884

Keypoint for Code for Print Result | 输出结果的关键点

- sort_values函数的应用

Summary for Demographic Filtering | 人工统计学的总结

```
Hurray! We have made our first(though very basic) recommender. Under the Trending Now tab of these systems we find movies that are very popular and they can just be obtained by sorting the dataset by the popularity column.
```

```
万岁！我们做出了第一个（尽管是非常基本的）推荐器。我们已经做出了我们的第一个（尽管是非常基本的）推荐器。在这些系统的Trending Now标签下，我们找到了非常受欢迎的电影，它们可以通过对数据集的人气列进行排序而获得。
```

Visualization for Result of Demographic Filtering | 结果可视化

Code for Visualization for Result of Demographic Filtering | 可视化编程

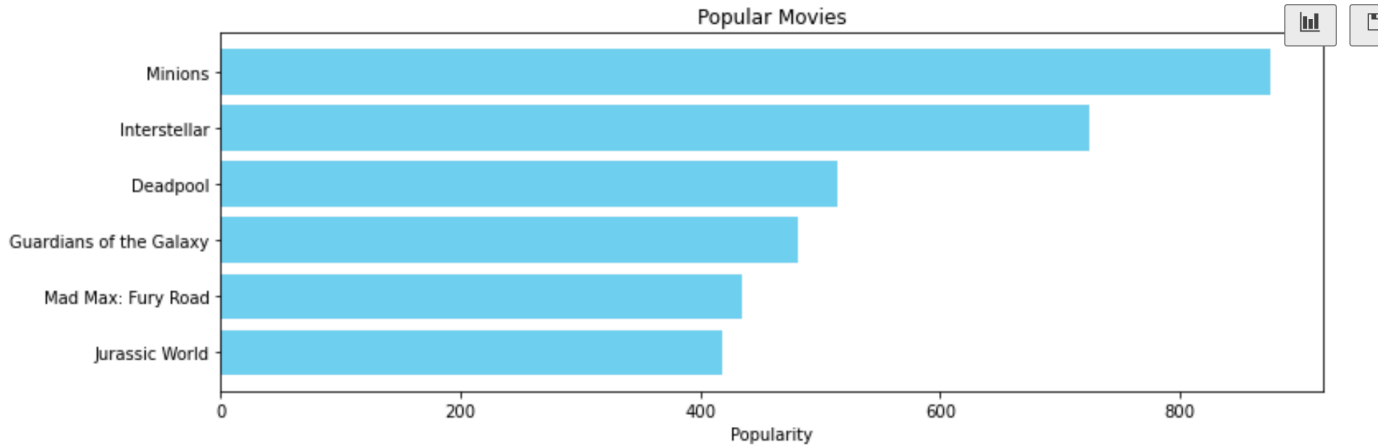
```
pop= df2.sort_values('popularity', ascending=False)
import matplotlib.pyplot as plt
plt.figure(figsize=(12,4))

plt.barh(pop['title'].head(6),pop['popularity'].head(6), align='center',
        color='skyblue')
plt.gca().invert_yaxis()
plt.xlabel("Popularity")
plt.title("Popular Movies")
```

```
This diagramming is not the focus of this course
```

```
作图不是本课程的重点
```

Result of Code for Visualization for Result of Demographic Filtering | 可视化的结果



Total Summary for Demographic Filtering | 人工统计学的总结

```
Now something to keep in mind is that these demographic recommender provide a
general chart of recommended movies to all the users. They are not sensitive to the
interests and tastes of a particular user. This is when we move on to a more refined
system- Content Basesd Filtering.
```

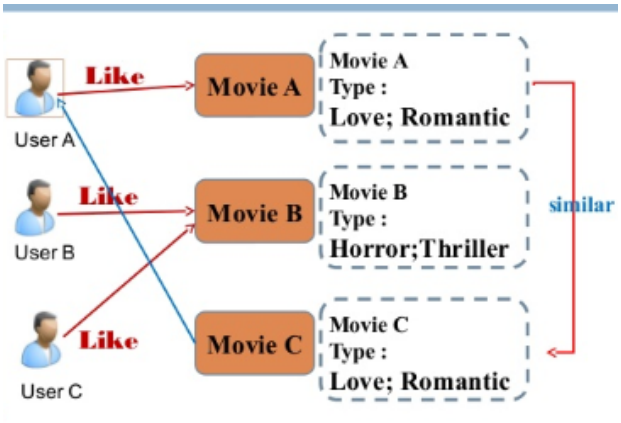
```
现在需要记住的是，这些人口统计学推荐器向所有用户提供一个推荐电影的一般图表。它们对特定用户的兴趣和品味不敏感。这时，我们就会转向一个更精细的系统--内容过滤。
```

Content Based Filtering | 基于内容的推荐

Introduction of Content Based Filtering | 基于内容的推荐的简介

```
In this recommender system the content of the movie (overview, cast, crew, keyword, tagline etc) is used to find its similarity with other movies. Then the movies that are most likely to be similar are recommended.
```

```
在这个推荐系统中，电影的内容（概述、演员、工作人员、关键词、标语等）被用来寻找它与其他电影的相似性。然后，最可能相似的电影被推荐。
```



Keypoint for Introduction of Content Based Filtering | 基于内容的推荐的简介的关键点

- 寻找几部电影的关键点（或称为相似性）从而从一部电影推荐像另一部电影

Plot description based Recommender | 基于情节描述的推荐程序

Introduction of Plot description based Recommender | 给予情节的推荐程序的简介

```
We will compute pairwise similarity scores for all movies based on their plot descriptions and recommend movies based on that similarity score. The plot description is given in the overview feature of our dataset. Let's take a look at the data. ..
```

```
我们将根据所有电影的情节描述计算出成对的相似性分数，并根据该相似性分数推荐电影。情节描述在我们的数据集的概述(overall)特征中给出。让我们来看看这些数据。..
```

Keypoint for Introduction of Plot description based Recommender | 基于情节推荐程序简介的关键点

- 从情节描述(overall)获得两部电影的相似性 (最重要)
- 根据相似性得分推荐电影

Take a look at the overall | 看一下一半的情节

```
df2['overview'].head(5)
```

[13] ✓ 0.6s

```
0    In the 22nd century, a paraplegic Marine is di...
1    Captain Barbossa, long believed to be dead, ha...
2    A cryptic message from Bond’s past sends him o...
3    Following the death of District Attorney Harve...
4    John Carter is a war-weary, former military ca...
Name: overview, dtype: object
```

TF-IDF | 词频-逆文件频率

This algorithm will extract the words from the movie description that best describe the movie

这个算法将从电影描述中提取中最能描述这部电影的词语

```
Notice:Here involves long text, will use a paragraph of Chinese and a paragraph of English form
注意：这里涉及到长文本，将会使用一段英文原文，一段英文翻译
```

```
For any of you who has done even a bit of text processing before knows we need to convert the word vector of each overview. Now we'll compute Term Frequency-Inverse Document Frequency (TF-IDF) vectors for each overview.
对于任何一个以前做过哪怕一点点文本处理的人来说，都知道我们需要转换每个概览的词向量。现在我们将为每个概览进行 TF-IDF(Term Frequency-Inverse Document Frequency, 词频-逆文件频率).

(
是一种用于资讯检索与资讯探勘的常用加权技术。TF-IDF是一种统计方法，用以评估一字词对于一个文件集或一个语料库中的其中一份文件的重要程度。字词的重要性随着它在文件中出现的次数成正比增加，但同时会随着它在语料库中出现的频率成反比下降。
上述引用总结就是，一个词语在一篇文章中出现次数越多，同时在所有文档中出现次数越少，越能够代表该文章.)
)

Now if you are wondering what is term frequency , it is the relative frequency of a word in a document and is given as (term instances/total instances). Inverse Document Frequency is the relative count of documents containing the term is given as log(number of documents/documents with term) The overall importance of each word to the documents in which they appear is equal to TF * IDF
现在，如果你想知道什么是词频 ，它是一个词在文档中的相对频率，并被赋予（术语实例/总实例）。逆向文档频率是指包含该词的文档的相对计数，给出为对数（文档数量/包含该词的文档） 每个词对其出现的文档的总体重要性等于TF * IDF
```

但是，需要注意，一些通用的词语对于主题并没有太大的作用，反倒是一些出现频率较少的词才能够表达文章的主题，所以单纯使用是TF不合适的。权重的设计必须满足：一个词预测主题的能力越强，权重越大，反之，权重越小。所有统计的文章中，一些词只是在其中很少几篇文章中出现，那么这样的词对文章的主题的作用很大，这些词的权重应该设计的较大。IDF就是在完成这样的工作.

This will give you a matrix where each column represents a word in the overview vocabulary (all the words that appear in at least one document) and each row represents a movie, as before.This is done to reduce the importance of words that occur frequently in plot overviews and therefore, their significance in computing the final similarity score.

这将给你一个矩阵，其中每一列代表概述词汇中的一个词（所有至少出现在一个文档中的词），每一行代表一部电影，如前所述。这样做是为了减少在情节概述中频繁出现的词的重要性，因此，它们在计算最终相似性分数时的重要性。

Fortunately, scikit-learn gives you a built-in TfIdfVectorizer class that produces the TF-IDF matrix in a couple of lines. That's great, isn't it?

幸运的是，scikit-learn给了你一个内置的TfIdfVectorizer类，只需几行就能产生TF-IDF矩阵。这很好，不是吗?

Notice: Added more instructions on hair counting in the Chinese description

注意：在中文的说明中，增加更多的算法的说明

Notice: 部分TF-ID说明来源于:https://blog.csdn.net/zrc199021/article/details/53728499 由：zrc199021原创编辑

Part of the TF-ID description from:https://blog.csdn.net/zrc199021/article/details/53728499 Originally edited by:zrc199021

Keypoint of TF-IDF | 词频-逆文件频率的关键点

- 我们需要将电影描述的词转换为向量
- TF-IDF是一种统计方法，用以评估一字词对于一个文件集或一个语料库中的其中一份文件的重要程度。字词的重要性随着它在文件中出现的次数成正比增加，但同时会随着它在语料库中出现的频率成反比下降
- 一个词语在一篇文章中出现次数越多, 同时在所有文档中出现次数越少, 越能够代表该文章
- 逆向文档频率是指包含该词的文档的相对计数，给出为对数（文档数量/包含该词的文档） 每个词对其出现的文档的总体重要性等于TF * IDF
- 一些通用的词语对于主题并没有太大的作用, 反倒是一些出现频率较少的词才能够表达文章的主题, 所以单纯使用是TF不合适的。权重的设计必须满足：一个词预测主题的能力越强，权重越大，反之，权重越小。所有统计的文章中，一些词只是在其中很少几篇文章中出现，那么这样的词对文章的主题的作用很大，这些词的权重应该设计的较大。IDF就是在完成这样的工作.
- 使用SkLearn来完成TD-IDF

Code for TF-IDF Learning | 涉及到 TF-IDF的代码实现

```
Notice:By default, you have learned to install all the required packages and all the rest of the tools in the article
注意：文章作者已经默认您已经配置好一切所需要的工具包括Python和Python的Package,甚至可能需要的VSCode和Jupyter工具
```

```
#Import TfIdfVectorizer from scikit-learn | 从 sklearn 导入 TF-IDF 工具
from sklearn.feature_extraction.text import TfidfVectorizer

#Define a TF-IDF Vectorizer Object. Remove all english stop words such as 'the', 'a' | 定义函数
tfidf = TfidfVectorizer(stop_words='english')

#Replace NaN with an empty string | 删除所有的空格符号
df2['overview'] = df2['overview'].fillna('')

#Construct the required TF-IDF matrix by fitting and transforming the data |进行TF-IDF转换
tfidf_matrix = tfidf.fit_transform(df2['overview'])

#Output the shape of tfidf_matrix | 查看返回的数据
tfidf_matrix.shape
```

Keypoint of Code for TF-IDF | TF-IDF的代码实现

- SKlearn Package
- TF-IDF Usage in SKlearn

Result for Code for TF-IDF Learning | TF-IDF的代码实现的结果

Commandline Output for Result for Code for TF-IDF Learning | 命令行输出

```
(4803, 20978)
```

Interpretation of algorithm results of TF-IDF Learning | TF-IDF算法的结果的解释

```
We see that over 20,000 different words were used to describe the 4800 movies in our dataset.

With this matrix in hand, we can now compute a similarity score. There are several candidates for this; such as the euclidean, the Pearson and the cosine similarity scores. There is no right answer to which score is the best. Different scores work well in different scenarios and it is often a good idea to experiment with different metrics.

We will be using the cosine similarity to calculate a numeric quantity that denotes the similarity between two movies. We use the cosine similarity score since it is independent of magnitude and is relatively easy and fast to calculate. Mathematically, it is defined as follows:
```

```
我们看到，在我们的数据集中，有超过2万个不同的词被用来描述4800部电影。

有了这个矩阵，我们现在可以计算出一个相似度分数。这方面有几个候选者，如欧几里得、皮尔逊和余弦相似度分数。关于哪个分数是最好的，没有正确的答案。不同的分数在不同的情况下效果很好，用不同的指标进行实验通常是一个好主意。

我们将使用余弦相似度来计算一个数字量，表示两部电影之间的相似度。我们使用余弦相似度分数，因为它与大小无关，而且计算起来相对容易和快速。在数学上，它的定义如下。
```

$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}},$$

Keypoint of Interpretation of algorithm results of TF-IDF Learning ｜TF-IDF算法的结果的解释的关键点

- 有超过2万个不同的词被用来描述4800部电影
- the euclidean, the Pearson and the cosine similarity scores (几里得、皮尔逊和余弦相似度分数) 不同的分数在不同的实验不同，建议都用，在本次试验中用 the cosine similarity scores.

Calculate the cosine similarity score ｜ 计算余弦相似度分数

Code for Calculate the cosine similarity score ｜ 计算余弦相似度分数的代码实现

```
# Import linear_kernel | 导入计算余弦复杂度工具
from sklearn.metrics.pairwise import linear_kernel

# Compute the cosine similarity matrix |计算余弦复杂度
cosine_sim = linear_kernel(tfidf_matrix, tfidf_matrix)
```

Get the ten most similar movie titles of the entered movies | 获得与输入电影最相似的十个电影

```
We are going to define a function that takes in a movie title as an input and outputs a list of the 10 most similar movies. Firstly, for this, we need a reverse mapping of movie titles and DataFrame indices. In other words, we need a mechanism to identify the index of a movie in our metadata DataFrame, given its title.
```

```
我们要定义一个函数，将电影标题作为输入，并输出10部最相似电影的列表。首先，为此，我们需要一个电影标题和DataFrame索引的反向映射。换句话说，我们需要一个机制来识别元数据DataFrame中的电影索引，给定其标题。
```

Keypoint of Get the ten most similar movie titles of the entered movies | 关于获得十个最相似电影的关键点

- 需要一个推荐函数 在给定一部电影的情况下能推荐十部最相似的电影
- 1.建立电影名字到序号的映射

Code Implementation for a reverse mapping of movie titles and DataFrame indices | 建立电影名字到标号的反向映射

```
#Construct a reverse map of indices and movie titles | 建立一个字典 用于将电影名字 -> 序号
indices = pd.Series(df2.index, index=df2['title']).drop_duplicates()
```

Introduction of Recommended function code implementation | 关于推荐函数的简介

```
We are now in a good position to define our recommendation function. These are the following steps we'll follow :-

Get the index of the movie given its title.
Get the list of cosine similarity scores for that particular movie with all movies. Convert it into a list of tuples where the first element is its position and the second is the similarity score.
Sort the aforementioned list of tuples based on the similarity scores; that is, the second element.
Get the top 10 elements of this list. Ignore the first element as it refers to self (the movie most similar to a particular movie is the movie itself).
Return the titles corresponding to the indices of the top elements.
```

Steps to create a recommendation function | 建立推荐函数的步骤

> 我们现在处于一个很好的位置来定义我们的推荐函数。以下是我们将遵循的步骤：

1. 获取电影标题的索引。 | 使用刚刚创建的表格
2. 获取该特定电影与所有电影的余弦相似度分数列表。将其转换成一个图元列表，其中第一个元素是其位置，第二个元素是相似度分数。
3. 根据相似度分数对上述图元列表进行排序；也就是说，第二个元素。
4. 得到这个列表中的前10个元素。忽略第一个元素，因为它指的是自己（与某部电影最相似的电影就是这部电影本身）。
5. 返回顶部元素的索引所对应的标题。

Code for Recommended function | 推荐函数的代码实现

```
# Function that takes in movie title as input and outputs most similar movies | 输入电影的标题 输出最相似的电影
def get_recommendations(title, cosine_sim=cosine_sim):
    # Get the index of the movie that matches the title | 通过电影标题反向索引到下标
    idx = indices[title]

    # Get the pairwsie similarity scores of all movies with that movie | 获得该电影的相似度分数列表
    sim_scores = list(enumerate(cosine_sim[idx]))

    # Sort the movies based on the similarity scores | 对其进行排序
    sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)

    # Get the scores of the 10 most similar movies | 获得 第二个到第十个 （第一个是自己不算）
    sim_scores = sim_scores[1:11]

    # Get the movie indices | 获得这些电影的下标
    movie_indices = [i[0] for i in sim_scores]

    # Return the top 10 most similar movies | 返回电影的名字
    return df2['title'].iloc[movie_indices]
```

Result For Code for Recommended function | 推荐函数的结果

```
get_recommendations('The Dark Knight Rises')
```

✓ 0.3s

```
65                              The Dark Knight
299                              Batman Forever
428                              Batman Returns
1359                                    Batman
3854    Batman: The Dark Knight Returns, Part 2
119                              Batman Begins
2507                                 Slow Burn
9           Batman v Superman: Dawn of Justice
1181                                      JFK
210                              Batman & Robin
Name: title, dtype: object
```

Summary about the content-based recommendation process | 关于基于内容推荐器的总结

```
While our system has done a decent job of finding movies with similar plot descriptions, the quality of recommendations is not that great. "The Dark Knight Rises" returns all Batman movies while it is more likely that the people who liked that movie are more inclined to enjoy other Christopher Nolan movies. This is something that cannot be captured by the present system.
```

```
虽然我们的系统在寻找具有类似情节描述的电影方面做得不错，但推荐的质量并不高。"黑暗骑士崛起 "返回所有的蝙蝠侠电影，而喜欢这部电影的人更有可能喜欢克里斯托弗-诺兰的其他电影。这是目前的系统所不能捕捉到的。
```

Credits, Genres and Keywords Based Recommender | 基于评分，类型和关键词的推荐

Introduction of Credits, Genres and Keywords Based Recommender | 基于评分，类型和关键词的推荐的简介

```
It goes without saying that the quality of our recommender would be increased with the usage of better metadata. That is exactly what we are going to do in this section. We are going to build a recommender based on the following metadata: the 3 top actors, the director, related genres and the movie plot keywords.

From the cast, crew and keywords features, we need to extract the three most important actors, the director and the keywords associated with that movie. Right now, our data is present in the form of "stringified" lists , we need to convert it into a safe and usable structure
```

```
不言而喻，如果使用更好的元数据，我们的推荐器的质量将得到提高。这正是我们在本节中要做的。我们将根据以下元数据建立一个推荐器：3个顶级演员、导演、相关类型和电影情节的关键词。

从演员、工作人员和关键词的特征中，我们需要提取三个最重要的演员、导演和与该电影相关的关键词。现在，我们的数据是以 "字符串化 "列表的形式出现的，我们需要将其转换为一个安全和可用的结构
```

Keypoint of Introduction of Credits, Genres and Keywords Based Recommender | 基于评分，类型和关键词的推荐的关键点

- 基于3个顶级演员（主演），导演，相关类型，电影情节的推荐器
- 我们需要提取csv文件中的以键值队形式存在的演员信息然后转换

Code for Extracting staff information from strings | 从字符串提取导演主演等信息的代码实现

Step 1. Parsing strings to Python objects | 将字符串转换为Python对象

```python
# Parse the stringified features into their corresponding python objects | 将字符串化的特征解析为相应的python对象
from ast import literal_eval

features = ['cast', 'crew', 'keywords', 'genres']
for feature in features:
    df2[feature] = df2[feature].apply(literal_eval)
```

Step 2. Get director function | 获得导演信息

```python
# Get the director's name from the crew feature. If director is not listed, return NaN
def get_director(x):
    for i in x:
        if i['job'] == 'Director':
            return i['name']
    return np.nan
```

Step 3. Get information on the first three lead actors | 获得前三个主演信息

```
# Returns the list top 3 elements or entire list; whichever is more.
def get_list(x):
    if isinstance(x, list):
        names = [i['name'] for i in x]
        #Check if more than 3 elements exist. If yes, return only first three. If no, return entire list.
        if len(names) > 3:
            names = names[:3]
        return names

    #Return empty list in case of missing/malformed data
    return []
```

Step 4. Pass the parsed data into df2 | 将信息传入df2

```
# Define new director, cast, genres and keywords features that are in a suitable form.
df2['director'] = df2['crew'].apply(get_director)

features = ['cast', 'keywords', 'genres']
for feature in features:
    df2[feature] = df2[feature].apply(get_list)
```

Step 5. Attempt to print information | 尝试打印信息

```
# Print the new features of the first 3 films
df2[['title', 'cast', 'director', 'keywords', 'genres']].head(3)
```

Result of Attempt to Print infomation

	title	cast	director	keywords	genres
0	Avatar	[Sam Worthington, Zoe Saldana, Sigourney Weaver]	James Cameron	[culture clash, future, space war]	[Action, Adventure, Fantasy]
1	Pirates of the Caribbean: At World's End	[Johnny Depp, Orlando Bloom, Keira Knightley]	Gore Verbinski	[ocean, drug abuse, exotic island]	[Adventure, Fantasy, Action]
2	Spectre	[Daniel Craig, Christoph Waltz, Léa Seydoux]	Sam Mendes	[spy, based on novel, secret agent]	[Action, Adventure, Crime]

Data Clean (Turn to lowercase and free of space) | 数据清洗，转小写，去除空格

```
The next step would be to convert the names and keyword instances into lowercase and strip all the spaces between them. This is done so that our vectorizer doesn't count the Johnny of "Johnny Depp" and "Johnny Galecki" as the same.
```

```
下一步将是把名字和关键词的实例转换为小写，并剥离它们之间的所有空格。这样做是为了让我们的向量器不把"Johnny Depp "和 "Johnny Galecki "的Johnny算作是同一个人。
```

Keypoint of Data Clean (Turn to lowercase and free of space) | 数据清洗的关键点

- 转小写
- 去空格

Code For Data Clean (Turn to lowercase and free of space) | 数据清洗的代码实现

Step 1. Constructing data cleaning functions | 构造数据清洗函数

```
# Function to convert all strings to lower case and strip names of spaces
def clean_data(x):
    if isinstance(x, list):
        return [str.lower(i.replace(" ", "")) for i in x]
    else:
        #Check if director exists. If not, return empty string
        if isinstance(x, str):
            return str.lower(x.replace(" ", ""))
        else:
            return ''
```

Step 2. Apply data cleaning functions | 应用数据清洗函数

```
# Apply clean_data function to your features.
features = ['cast', 'keywords', 'director', 'genres']

for feature in features:
    df2[feature] = df2[feature].apply(clean_data)
```

Keypoint of Code for Data Clean (Turn to lowercase and free of space) | 数据清洗的代码实现的关键点

- Isinstance() 函数

Create Metadata Soup | 创建原数据汤？

```
We are now in a position to create our "metadata soup", which is a string that
contains all the metadata that we want to feed to our vectorizer (namely actors,
director and keywords).
```

```
我们现在可以创建我们的 "元数据汤"，它是一个字符串，包含了我们想要提供给矢量器的所有元数据（即演员、
导演和关键词）。
```

Code for Create Metadata Soup

```
def create_soup(x):
    return ' '.join(x['keywords']) + ' ' + ' '.join(x['cast']) + ' ' + x['director'] + ' ' + ' '.join(x['genres'])
df2['soup'] = df2.apply(create_soup, axis=1)
```

Generate recommendation functions | 生成推荐函数

Introduction of Generate recommendation functions | 生成推荐函数简介

```
The next steps are the same as what we did with our plot description based recommender. One important difference is that we use the CountVectorizer() instead of TF-IDF. This is because we do not want to down-weight the presence of an actor/director if he or she has acted or directed in relatively more movies. It doesn't make much intuitive sense.
```

```
接下来的步骤与我们在基于情节描述的推荐器中所做的相同。一个重要的区别是，我们使用CountVectorizer()而不是TF-IDF。这是因为，如果一个演员/导演在相对较多的电影中担任过演员或导演，我们不想降低其存在的权重。这并没有什么直观的意义。
```

Keypoint of Introduction of Generate recommendation functions | 生成推荐函数简介的关键点

- 采用 CountVectorizer() 而非原来的算法，因为导演和主演的多次重复出现，并不影响权重

Code for Generate recommendation functions | 建立推荐函数

Step 1. Create the count matrix | 创建 count_matrix

```
# Import CountVectorizer and create the count matrix | 导入CountVectorizer函数并创建计数矩阵。
from sklearn.feature_extraction.text import CountVectorizer

count = CountVectorizer(stop_words='english')
count_matrix = count.fit_transform(df2['soup'])
```

step 2. Compute the Cosine Similarity matrix | 计算余弦相似度

```
# Compute the Cosine Similarity matrix based on the count_matrix | 根据计数矩阵的值计算余弦相似度
from sklearn.metrics.pairwise import cosine_similarity

cosine_sim2 = cosine_similarity(count_matrix, count_matrix)
```

Step 3. Construct reverse mapping | 建立反向索引

```
# Reset index of our main DataFrame and construct reverse mapping as before | 创建反向索引
df2 = df2.reset_index()
indices = pd.Series(df2.index, index=df2['title'])
```

Keypoint of Code for Generate recommendation functions | 创建推荐函数的代码实现的关键点

- 与ContentBased类似的使用CountVectorizer函数转化
- 一样计算余弦相似度
- 同样进行反向索引

Result of Credits, Genres and Keywords Based Recommender | 基于评分，类型和关键词的推荐的结果

```
get_recommendations('The Dark Knight Rises', cosine_sim2)
```

✓ 0.3s

```
65               The Dark Knight
119                Batman Begins
4638    Amidst the Devil's Wings
1196                The Prestige
3073           Romeo Is Bleeding
3326               Black November
1503                       Takers
1986                       Faster
303                      Catwoman
747                Gangster Squad
Name: title, dtype: object
```

Summary for Result of Credits, Genres and Keywords Based Recommender | 基于评分，类型和关键词的推荐的总结

```
We see that our recommender has been successful in capturing more information due to more metadata and has given us (arguably) better recommendations. It is more likely that Marvels or DC comics fans will like the movies of the same production house. Therefore, to our features above we can add production_company . We can also increase the weight of the director , by adding the feature multiple times in the soup.
```

```
我们看到，由于元数据较多，我们的推荐器成功地捕捉到了更多的信息，并给了我们（可以说）更好的推荐。惊叹号或DC漫画的粉丝更有可能喜欢同一制作公司的电影。因此，在我们的上述特征中，我们可以增加 production_company 。我们还可以增加导演的权重，通过在汤中多次添加该特征。
```

Collaborative Filtering | 协同过滤

Introduction of Collaborative Filtering | 协同过滤的简介

Shortage of Recommender Function Above | 前面的推荐器的短板

```
Our content based engine suffers from some severe limitations. It is only capable of suggesting movies which are close to a certain movie. That is, it is not capable of capturing tastes and providing recommendations across genres.
```

```
我们基于内容的引擎受到一些严重的限制。它只能够推荐与某部电影接近的电影。也就是说，它没有能力捕捉品味并提供跨类型的推荐。
```

```
Also, the engine that we built is not really personal in that it doesn't capture the personal tastes and biases of a user. Anyone querying our engine for recommendations based on a movie will receive the same recommendations for that movie, regardless of who she/he is.
```

```
此外，我们建立的引擎并不是真正的个人化，因为它没有捕捉到用户的个人品味和偏见。任何查询我们的引擎以获得基于电影的推荐的人，都会收到关于该电影的相同推荐，无论她/他是谁。
```

Summary above

- 只能推荐与当前内容的接近内容，而不能推荐跨度较大的内容
- 与用户的选择无关，任何人在观看这部影片的时候都会获得相同的推荐 （或者可以说与单个用户无关）

Collaborative Filtering | 协作过滤

```
Therefore, in this section, we will use a technique called Collaborative Filtering to make recommendations to Movie Watchers. It is basically of two types:-
```

因此，在本节中，我们将使用一种叫做协作过滤的技术来向电影观看者进行推荐。它基本上有两种类型：-

User based filtering- These systems recommend products to a user that similar users have liked. For measuring the similarity between two users we can either use pearson correlation or cosine similarity. This filtering technique can be illustrated with an example. In the following matrixes, each row represents a user, while the columns correspond to different movies except the last one which records the similarity between that user and the target user. Each cell represents the rating that the user gives to that movie. Assume user E is the target.

基于用户的过滤--这些系统向用户推荐类似用户喜欢的产品。为了测量两个用户之间的相似度，我们可以使用佩尔森相关或余弦相似。这种过滤技术可以用一个例子来说明。在下面的矩阵中，每一行代表一个用户，而每一列对应不同的电影，除了最后一列记录该用户和目标用户之间的相似度。每个单元格代表用户对该电影的评价。假设用户E是目标用户。

	The Avengers	Sherlock	Transformers	Matrix	Titanic	Me Before You	Similarity(i, E)
A	2		2	4	5		NA
B	5		4			1	
C			5		2		
D		1		5		4	
E			4			2	1
F	4	5		1			NA

Since user A and F do not share any movie ratings in common with user E, their similarities with user E are not defined in Pearson Correlation. Therefore, we only need to consider user B, C, and D. Based on Pearson Correlation, we can compute the following similarity.

由于用户A和F与用户E没有任何共同的电影评分，他们与用户E的相似性没有在Pearson Correlation中定义。因此，我们只需要考虑用户B、C和D。基于Pearson Correlation，我们可以计算出以下相似度。

	The Avengers	Sherlock	Transformers	Matrix	Titanic	Me Before You	Similarity(i, E)
A	2		2	4	5		NA
B	5		4			1	0.87
C			5		2		1
D		1		5		4	-1
E			4			2	1
F	4	5		1			NA

```
From the above table we can see that user D is very different from user E as the Pearson Correlation between them is negative. He rated Me Before You higher than his rating average, while user E did the opposite. Now, we can start to fill in the blank for the movies that user E has not rated based on other users.
```

```
从上表我们可以看出，用户D与用户E非常不同，因为他们之间的Pearson Correlation是负的。他对《Me Before You》的评分高于他的评分平均值，而用户E则相反。现在，我们可以开始根据其他用户的情况，为用户E没有评分的电影填空。
```

	The Avengers	Sherlock	Transformers	Matrix	Titanic	Me Before You	Similarity(i, E)
A	2		2	4	5		NA
B	5		4			1	0.87
C			5		2		1
D		1		5		4	-1
E	3.51*	3.81*	4	2.42*	2.48*	2	1
F	4	5		1			NA

```
Although computing user-based CF is very simple, it suffers from several problems. One main issue is that users' preference can change over time. It indicates that precomputing the matrix based on their neighboring users may lead to bad performance. To tackle this problem, we can apply item-based CF.
```

```
虽然计算基于用户的CF非常简单，但它存在几个问题。一个主要问题是，用户的偏好会随着时间的推移而改变。这表明，基于他们的邻居用户预先计算矩阵可能会导致糟糕的性能。为了解决这个问题，我们可以应用基于项目的CF。
```

Item Based Collaborative Filtering | 基于项目的协同过滤

Instead of measuring the similarity between users, the item-based CF recommends items based on their similarity with the items that the target user rated. Likewise, the similarity can be computed with Pearson Correlation or Cosine Similarity. The major difference is that, with item-based collaborative filtering, we fill in the blank vertically, as oppose to the horizontal manner that user-based CF does. The following table shows how to do so for the movie Me Before You.

--基于项目的协同过滤不是测量用户之间的相似性，而是根据它们与目标用户评价的项目的相似性来推荐项目。同样地，相似度可以用皮尔逊相关或余弦相似来计算。主要的区别是，在基于项目的协同过滤中，我们是纵向填空的，而不是基于用户的CF的横向填空方式。下表显示了如何对电影《我在你之前》进行填写。

	The Avengers	Sherlock	Transformers	Matrix	Titanic	Me Before You
A	2		2	4	5	2.94*
B	5		4			1
C			5		2	2.48*
D		1		5		4
E			4			2
F	4	5		1		1.12*
Similarity	-1	-1	0.86	1	1	

It successfully avoids the problem posed by dynamic user preference as item-based CF is more static. However, several problems remain for this method. First, the main issue is scalability. The computation grows with both the customer and the product. The worst case complexity is O(mn) with m users and n items. In addition, sparsity is another concern. Take a look at the above table again. Although there is only one user that rated both Matrix and Titanic rated, the similarity between them is 1. In extreme cases, we can have millions of users and the similarity between two fairly different movies could be very high simply because they have similar rank for the only user who ranked them both.

它成功地避免了动态用户偏好所带来的问题，因为基于项目的CF是比较静态的。然而，这种方法仍然存在几个问题。首先，主要问题是可扩展性。计算量随着客户和产品的增加而增加。最坏的情况是，在有m个用户和n个项目的情况下，复杂度为O(mn)。此外，稀疏性是另一个问题。再看一下上面的表格。虽然只有一个用户对《黑客帝国》和《泰坦尼克号》进行了评分，但它们之间的相似度是1。在极端情况下，我们可能有数百万用户，两部相当不同的电影之间的相似度可能非常高，仅仅是因为它们在唯一一个对它们进行排名的用户那里有相似的排名。

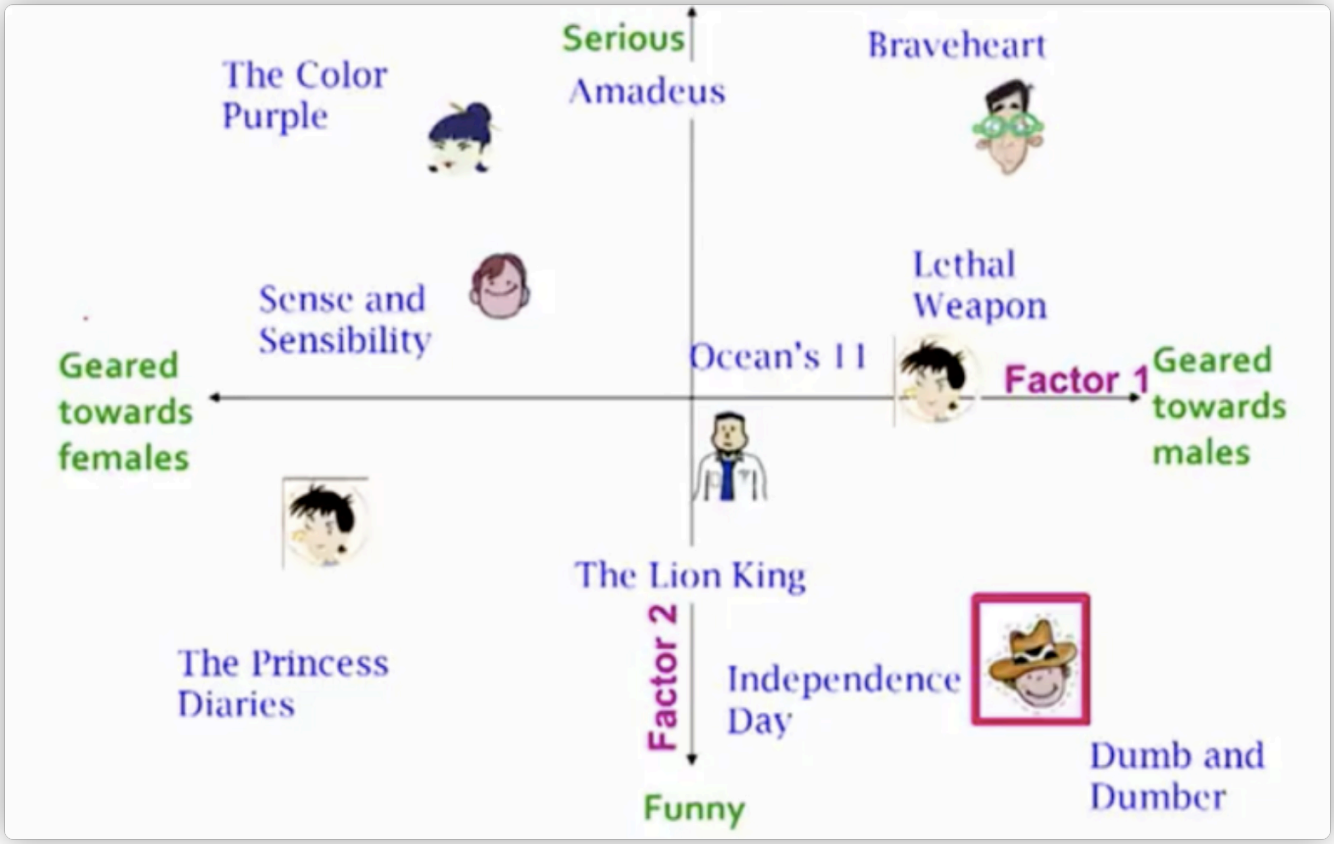
Single Value Decomposition | 奇异值分解

One way to handle the scalability and sparsity issue created by CF is to leverage a latent factor model to capture the similarity between users and items. Essentially, we want to turn the recommendation problem into an optimization problem. We can view it as how good we are in predicting the rating for items given a user. One common metric is Root Mean Square Error (RMSE). The lower the RMSE, the better the performance.

处理CF所产生的可扩展性和稀疏性问题的一种方法是利用潜在的因素模型来捕捉用户和项目之间的相似性。从本质上讲，我们想把推荐问题变成一个优化问题。我们可以把它看作是我们在预测给定用户的项目的评级方面有多好。一个常见的指标是均方根误差（RMSE）。RMSE越低，性能就越好。

Now talking about latent factor you might be wondering what is it ?It is a broad idea which describes a property or concept that a user or an item have. For instance, for music, latent factor can refer to the genre that the music belongs to. SVD decreases the dimension of the utility matrix by extracting its latent factors. Essentially, we map each user and each item into a latent space with dimension r. Therefore, it helps us better understand the relationship between users and items as they become directly comparable. The below figure illustrates this idea.

现在谈到潜在因素，你可能想知道它是什么？它是一个广泛的概念，描述了一个用户或一个项目的属性或概念。例如，对于音乐来说，潜在因素可以指的是音乐所属的流派。SVD通过提取潜在因素来降低效用矩阵的维度。本质上，我们将每个用户和每个项目映射到一个维度为r的潜在空间。因此，它帮助我们更好地理解用户和项目之间的关系，因为它们变得直接可比。下图说明了这个想法。



```
Now enough said , let's see how to implement this. Since the dataset we used before did not have userId(which is necessary for collaborative filtering) let's load another dataset. We'll be using the Surprise library to implement SVD.
```

```
现在说得够多了，让我们看看如何实现这一点。由于我们之前使用的数据集没有userId（这是协同过滤所必需的），我们来加载另一个数据集。我们将使用Surprise库来实现SVD。
```

Code for Collaborative Filtering | 协作过滤的代码实现

```
Notice :
According to the official documentation, the evaluate() method was deprecated in version 1.0.5 (functionally replaced by model_selection.cross_validate()) and removed in version 1.1.0 (probably installed)
All source code that uses the evaluate method will be replaced with cross_validate
Please note: the code here will be different from Kaggle!
If you don't know how to replace it, please use Kaggle's official Notebook to learn
```

```
注意：
根据官方文档，evaluate()方法在1.0.5版中已弃用（功能上已由model_selection.cross_validate()取代），并在1.1.0版中删除（可能已安装）
所有源代码用到evaluate方法都将会被替换为cross_validate
请注意：这里的代码会与Kaggle不同！
如果不会替换，请使用Kaggle的官方Notebook学习
```

Step 1. Importing datasets and library files | 导入数据集合和库文件

```
from surprise import Reader, Dataset, SVD # evaluate
from surprise.model_selection import cross_validate # 这里替换为原来的 evaluate
reader = Reader()
ratings = pd.read_csv('/Users/llonvne/OneDrive/大数据挖掘/大作业/电影推荐入门/dataset/dataset/ratings_small.csv')
ratings.head()
```

```
Note that in this dataset movies are rated on a scale of 5 unlike the earlier one.
```

```
请注意，在这个数据集中，电影的评分标准是5分，与之前的不同。
```

```
data = Dataset.load_from_df(ratings[['userId', 'movieId', 'rating']], reader)
```

Step 2. Do a five-layer fold repeat experiment | 五层折叠重复实验

```
svd = SVD()
cross_validate(svd, data, measures=['RMSE', 'MAE'],cv=5) # 原来这里是evaluate函数更改一下，然后再这里设置五重折叠实验 (cv = 5)
```

Commandline Ouput for Step 2. Do a five-layer fold repeat experiment | 五层折叠实验结果

```
{'test_rmse': array([0.89365221, 0.88579453, 0.89633659, 0.89411305, 0.88788093,
        0.89781664, 0.88869267, 0.89270588, 0.88712254, 0.88673704]),
 'test_mae': array([0.68556129, 0.68337576, 0.68927964, 0.68681439, 0.68263146,
        0.69150679, 0.68236268, 0.68348932, 0.68268937, 0.68034705]),
 'fit_time': (4.438669919967651,
  4.458031177520752,
  4.409940958023071,
  4.4417033195495605,
  4.466619253158569,
  4.451452732086182,
  4.436136960983276,
  4.507695198059082,
  4.6954309940338135,
  4.593163013458252),
 'test_time': (0.03911900520324707,
  0.039916038513183594,
  0.10521388053894043,
  0.03905487060546875,
  0.039415836334228516,
  0.04060482978820801,
  0.03899788856506348,
  0.04242277145385742,
  0.04042696952819824,
  0.039530038833618164)}
```

```
We get a mean Root Mean Sqaure Error of 0.89 approx which is more than good enough for our case. Let us now train on our dataset and arrive at predictions.
```

```
我们得到的平均均方根误差为0.89左右，这对我们的情况来说已经足够好了。现在让我们对我们的数据集进行训练并得出预测结果。
```

Step 3. Fit Trainset | 训练测试数据

```
trainset = data.build_full_trainset()
svd.fit(trainset)
```

Predict users' interest in any movie ｜ 预测用户对任一的兴趣值

Code for Predict users' interest in any movie ｜ 预测用户

```
svd.predict(1, 302, 3) #
```

Result of Code for Predict users' interest in any movie | 预测用户的兴趣值的结果

```
Prediction(uid=1, iid=302, r_ui=3, est=2.754950316852029, details={'was_impossible': False})
```

```
对于 ID 为 1 的用户 对于 ID 为 302 的电影预测值为 2.618
```

```
For movie with ID 302, we get an estimated prediction of 2.618. One startling feature of this recommender system is that it doesn't care what the movie is (or what it contains). It works purely on the basis of an assigned movie ID and tries to predict ratings based on how the other users have predicted the movie.
```

```
对于ID为302的电影，我们得到的估计预测值为2.618。这个推荐系统的一个惊人的特点是，它并不关心电影是什么（或它包含什么）。它纯粹是根据一个指定的电影ID来工作，并试图根据其他用户对电影的预测来预测评分。
```

Keypoint of Predict users' interest in any movie ｜ 预测的关键点

- 它并不关心电影是什么（或它包含什么）。它纯粹是根据一个指定的电影ID来工作，并试图根据其他用户对电影的预测来预测评分。

Original Article Summary ｜ 原始文章的总结

```
We create recommenders using demographic , content- based and collaborative filtering. While demographic filtering is very elemantary and cannot be used practically, Hybrid Systems can take advantage of content-based and collaborative filtering as the two approaches are proved to be almost complimentary. This model was very baseline and only provides a fundamental framework to start with.
```

```
我们使用人口统计、基于内容和协作过滤来创建推荐器。虽然人口统计学过滤是非常重要的，不能实际使用，但混合系统可以利用基于内容和协作过滤的优势，因为这两种方法被证明是几乎互补的。这个模型是非常基线的，只提供了一个基本的框架来开始。
```

Total Article Summary ｜ 总结

```
在本次学习Kaggle文章中，我学会了最基本的基于人口的推荐器，学会了简单的基于内容的推荐器和基于评分关键字的推荐器，再后来我又学习如何使用协同过滤的方法，在其中我又学会了如何导入，清洗数据，怎么处理数据，和关于大数据的各种算法
```