奇异值分解（SVD）与推荐系统

目录 ¶

学习目标

- 了解奇异值分解的原理及推导过程。
- 能在推荐算法中应用SVD解决问题。

一、奇异值分解（SVD）

1 回顾特征值和特征向量

首先回顾下特征值和特征向量的定义如下：

$$Ax = \lambda x$$

其中A是一个n×n矩阵，x是一个n维向量，则λ是矩阵A的一个特征值，而x是矩阵A的特征值λ所对应的特征向量。

求出特征值和特征向量有什么好处呢？ 就是我们可以将矩阵A特征分解。如果我们求出了矩阵A的n个特征值$\lambda_1 \leqslant \lambda_2 \leqslant ... \leqslant \lambda_3$，以及这n个特征值所对应的特征向量$w_1, w_2, ... w_n$，那么矩阵A就可以用下式的特征分解表示：

$$A = W\Sigma W^{-1}$$

其中W是这n个特征向量所张成的n×n维矩阵，而Σ为这n个特征值为主对角线的n×n维矩阵。一般我们会把W的这n个特征向量标准化，即满足$||w_i||_2 = 1$，或者$w_i^T w_i = 1$，此时W的n个特征向量为标准正交基，满足$W^T W = I$，即$W^T = W^{-1}$，也就是说W为酉矩阵。 这样我们的特征分解表达式可以写成:

$$A = W\Sigma W^T$$

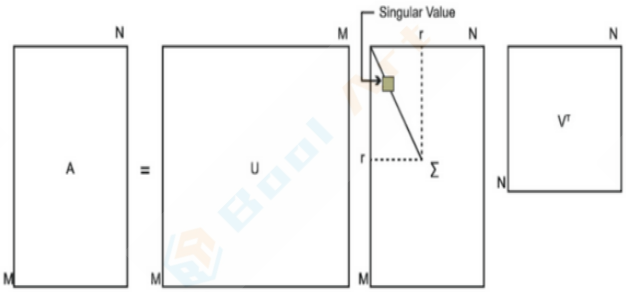
注意到要进行特征分解，矩阵A必须为方阵。那么如果A不是方阵，即行和列不相同时，我们还可以对矩阵进行分解吗？答案是可以，此时我们的SVD登场了。

2 SVD的定义

SVD也是对矩阵进行分解，但是和特征分解不同，SVD并不要求要分解的矩阵为方阵。假设我们的矩阵A是一个m×n的矩阵，那么我们定义矩阵A的SVD为：

$$A = U\Sigma V^T$$

其中U是一个m×m的矩阵，Σ是一个m×n的矩阵，除了主对角线上的元素以外全为0，主对角线上的每个元素都称为奇异值，V是一个n×n的矩阵。U和V都是酉矩阵，即满足$U^T U = I$，$V^T V = I$，下图可以很形象的看出上面SVD的定义：



那么我们如何求出SVD分解后的U,Σ,V这三个矩阵呢?

如果我们将A的转置和A做矩阵乘法，那么会得到n×n的一个方阵$A^T A$。既然$A^T A$是方阵，那么我们就可以进行特征分解，得到的特征值和特征向量满足下式：

$$(A^T A)v_i = \lambda_i v_i$$

这样我们就可以得到矩阵$A^T A$的n个特征值和对应的n个特征向量v了。将$A^T A$的所有特征向量张成一个n×n的矩阵V，就是我们SVD公式里面的V矩阵了。一般我们将V中的每个特征向量叫做A的右奇异向量。 如果我们将A和A的转置做矩阵乘法，那么会得到m×m的一个方阵AA^T。既然AA^T是方阵，那么我们就可以进行特征分解，得到的特征值和特征向量满足下式：

$$(AA^T)u_i = \lambda_i u_i$$

这样我们就可以得到矩阵AA^T的m个特征值和对应的m个特征向量u了。将AA^T的所有特征向量张成一个m×m的矩阵U，就是我们SVD公式里面的U矩阵了。一般我们将U中的每个特征向量叫做A的左奇异向量。

U和V我们都求出来了，现在就剩下奇异值矩阵Σ没有求出了。 由于Σ除了对角线上是奇异值其他位置都是0，那我们只需要求出每个奇异值σ就可以了。 我们注意到:

$$A = U\Sigma V^T \Rightarrow AV = U\Sigma V^T V \Rightarrow AV = U\Sigma \Rightarrow Av_i = \sigma_i u_i \Rightarrow \sigma_i = Av_i / u_i$$

这样我们可以求出我们的每个奇异值，进而求出奇异值矩阵Σ。

上面还有一个问题没有讲，就是我们说A^TA的特征向量组成的就是我们SVD中的V矩阵，而AA^T的特征向量组成的就是我们SVD中的U矩阵，这有什么根据吗？这个其实很容易证明，我们以V矩阵的证明为例。

$$A = U\Sigma V^T \Rightarrow A^T = V\Sigma U^T \Rightarrow A^TA = V\Sigma U^T U\Sigma V^T = V\Sigma^2 V^T$$

上式证明使用了$U^U = I, \Sigma^T = \Sigma$。可以看出$A^TA$的特征向量组成的的确就是我们SVD中的V矩阵。类似的方法可以得到AA^T的特征向量组成的就是我们SVD中的U矩阵。

进一步我们还可以看出我们的特征值矩阵等于奇异值矩阵的平方，也就是说特征值和奇异值满足如下关系：

$$\sigma_i = \sqrt{\lambda_i}$$

这样也就是说，我们可以不用$\sigma_i = \frac{Av_i}{u_i}$来计算奇异值，也可以通过求出$A^TA$的特征值取平方根来求奇异值。

3 SVD计算举例

这里我们用一个简单的例子来说明矩阵是如何进行奇异值分解的。我们的矩阵A定义为：

$$\mathbf{A} = \begin{pmatrix} 0 & 1 \\ 1 & 1 \\ 1 & 0 \end{pmatrix}$$

首先求出A^TA和AA^T

$$\mathbf{A^TA} = \begin{pmatrix} 0 & 1 & 1 \\ 1 & 1 & 0 \end{pmatrix} \begin{pmatrix} 0 & 1 \\ 1 & 1 \\ 1 & 0 \end{pmatrix} = \begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix}$$

$$\mathbf{AA^T} = \begin{pmatrix} 0 & 1 \\ 1 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 0 & 1 & 1 \\ 1 & 1 & 0 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 0 \\ 1 & 2 & 1 \\ 0 & 1 & 1 \end{pmatrix}$$

进而求出A^TA的特征值和特征向量：

$$\lambda_1 = 3; v_1 = \begin{pmatrix} 1/\sqrt{2} \\ 1/\sqrt{2} \end{pmatrix}; \lambda_2 = 1; v_2 = \begin{pmatrix} -1/\sqrt{2} \\ 1/\sqrt{2} \end{pmatrix}$$

接着求出AA^T的特征值和特征向量：

$$\lambda_1 = 3; u_1 = \begin{pmatrix} 1/\sqrt{6} \\ 2/\sqrt{6} \\ 1/\sqrt{6} \end{pmatrix}; \lambda_2 = 1; u_2 = \begin{pmatrix} 1/\sqrt{2} \\ 0 \\ -1/\sqrt{2} \end{pmatrix}; \lambda_3 = 0; u_3 = \begin{pmatrix} 1/\sqrt{3} \\ -1/\sqrt{3} \\ 1/\sqrt{3} \end{pmatrix}$$

利用$Av_i = \sigma_i, u_i, i = 1, 2$求奇异值：

$$\begin{pmatrix} 0 & 1 \\ 1 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 1/\sqrt{2} \\ 1/\sqrt{2} \end{pmatrix} = \sigma_1 \begin{pmatrix} 1/\sqrt{6} \\ 2/\sqrt{6} \\ 1/\sqrt{6} \end{pmatrix} \Rightarrow \sigma_1 = \sqrt{3}$$

$$\begin{pmatrix} 0 & 1 \\ 1 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} -1/\sqrt{2} \\ 1/\sqrt{2} \end{pmatrix} = \sigma_2 \begin{pmatrix} 1/\sqrt{2} \\ 0 \\ -1/\sqrt{2} \end{pmatrix} \Rightarrow \sigma_2 = 1$$

也可以用$\sigma_i = \sqrt{\lambda_i}$直接求出奇异值为$\sqrt{3}$和1. 最终得到A的奇异值分解为：

$$A = U\Sigma V^T = \begin{pmatrix} 1/\sqrt{6} & 1/\sqrt{2} & 1/\sqrt{3} \\ 2/\sqrt{6} & 0 & -1/\sqrt{3} \\ 1/\sqrt{6} & -1/\sqrt{2} & 1/\sqrt{3} \end{pmatrix} \begin{pmatrix} \sqrt{3} & 0 \\ 0 & 1 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} 1/\sqrt{2} & 1/\sqrt{2} \\ -1/\sqrt{2} & 1/\sqrt{2} \end{pmatrix}$$

4 SVD的一些性质

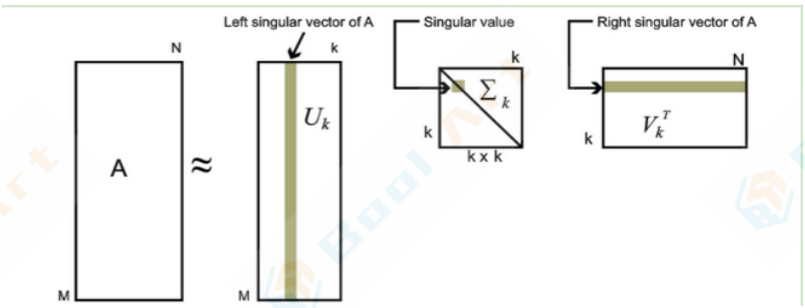
对于奇异值,它跟我们特征分解中的特征值类似，在奇异值矩阵中也是按照从大到小排列，而且奇异值的减少特别的快，在很多情况下，前10%甚至1%的奇异值的和就占了全部的奇异值之和的99%以上的比例。 也就是说，我们也可以用最大的k个的奇异值和对应的左右奇异向量来近似描述矩阵。 也就是说：

$$A_{m\times n} = U_{m\times m}\,\Sigma_{m\times n}\,V^T_{n\times n} \approx U_{m\times k}\,\Sigma_{k\times k}\,V^T_{k\times n}$$

其中k要比n小很多，也就是一个大的矩阵A可以用三个小的矩阵来表示。

$$U_{m\times k}, \sum_{k\times k}, V^T_{k\times n}$$

如下图所示，现在我们的矩阵A只需要灰色的部分的三个小矩阵就可以近似描述了。



由于这个重要的性质，SVD可以用于PCA降维，来做数据压缩和去噪。也可以用于推荐算法，将用户和喜好对应的矩阵做特征分解，进而得到隐含的用户需求来做推荐。同时也可以用于NLP中的算法，比如潜在语义索引（LSI）。

下面我们就对SVD用于PCA降维做一个介绍。

5 SVD用于PCA

PCA降维，需要找到样本协方差矩阵X^TX的最大的d个特征向量，然后用这最大的d个特征向量张成的矩阵来做低维投影降维。可以看出，在这个过程中需要先求出协方差矩阵X^TX，当样本数多样本特征数也多的时候，这个计算量是很大的。

注意到我们的SVD也可以得到协方差矩阵X^TX最大的d个特征向量张成的矩阵，但是SVD有个好处，有一些SVD的实现算法可以不求先求出协方差矩阵X^TX，也能求出我们的右奇异矩阵V。也就是说，我们的PCA算法可以不用做特征分解，而是做SVD来完成。这个方法在样本量很大的时候很有效。实际上，scikit-learn的PCA算法的背后真正的实现就是用的SVD，而不是我们我们认为的暴力特征分解。

另一方面，注意到PCA仅仅使用了我们SVD的右奇异矩阵，没有使用左奇异矩阵，那么左奇异矩阵有什么用呢?

假设我们的样本是m×n的矩阵X，如果我们通过SVD找到了矩阵XX^T最大的d个特征向量张成的m×d维矩阵U，则我们如果进行如下处理：

$$X'_{d\times n} = U^T_{d\times m} X_{m\times n}$$

可以得到一个d×n的矩阵X‘,这个矩阵和我们原来的m×n维样本矩阵X相比，行数从m减到了k，可见对行数进行了压缩。

- 左奇异矩阵可以用于行数的压缩。
- 右奇异矩阵可以用于列数即特征维度的压缩，也就是我们的PCA降维。

6 SVD小结

- SVD作为一个应用很广的算法，在很多机器学习算法中都有它的身影，在大数据中，由于SVD可以实现并行化，因此更是大展身手。
- SVD的缺点是分解出的矩阵解释性往往不强，有点黑盒子的味道，不过这不影响它的使用。

二、奇异值分解（SVD）的应用

可以帮助你选择机器学习算法和模型。

做机器学习会发现，当你拿到一个新的数据集时，选择适合它的算法和参数是很难的，一般都是靠着直觉和经验去做。然而效果并不好，而客观上说我们也没有足够的资源去尝试所有的可能性。对于大部分机器学习实践者而言，选择模型是个无解题。

换个问题。如果你是一个电商平台，此时来了个新客户，你怎么给她推荐商品?

仔细一想后你会发现，为一个新的数据选择适合的模型和参数，就跟给一个新用户推荐商品是一个意思？其实都是「在冷启动环境下的协同过滤」。也就是你对新数据（用户）几乎一无所知的前提下如何选择模型（推荐商品）。

那么这个时候，我们就可以用矩阵分解（奇异值分解可以来做矩阵分解）。

给定历史训练数据P，一个nxm的矩阵，此处我们有n个训练数据以及m个可选择的模型，P_{ij}代表第j个模型在第i个数据上的表现（比如准确度或者roc）。之后对P进行矩阵分解$P =< U, V^T >$来学习U和V矩阵，分别代表数据和模型在latent dimension（隐空间）上面的一些特点。

当一个新的数据集来的时候，你只需要先生成对应的U'，就可以通过$P' =< U', V^T > \in R^{1\times m}$来得到新的数据集在m个模型上的表现的预测了。

此处还需要考虑一点，如何在有一个新数据集时得到对应的U'。现在比较主流的方法是训练一个回归，直接从数据本身回归到对应的U'上去。比较常用的方法是对于所有的训练数据抽象为nxd的元特征（meta-features）矩阵M，生成的过程包括统计数据的长度，维度，分布等，之后再把元特征回归到学习到的U上面去，可以用随机森林做个非线性的多元回归。

为什么矩阵分解可以做模型选择？主要还是依靠分解过程去学习数据集本身的关联性，以及模型间的关联性，以及两者的联动。矩阵分解的过程是对数据的压缩，因此可以得到更加精炼的表示并去除掉一些干扰和异常。如果我们再有一个方法能将任一数据转化到矩阵U上，就可以简单的通过点乘来获得m个模型在新数据上的表现的预测，之后就可以选择最好的那个了。

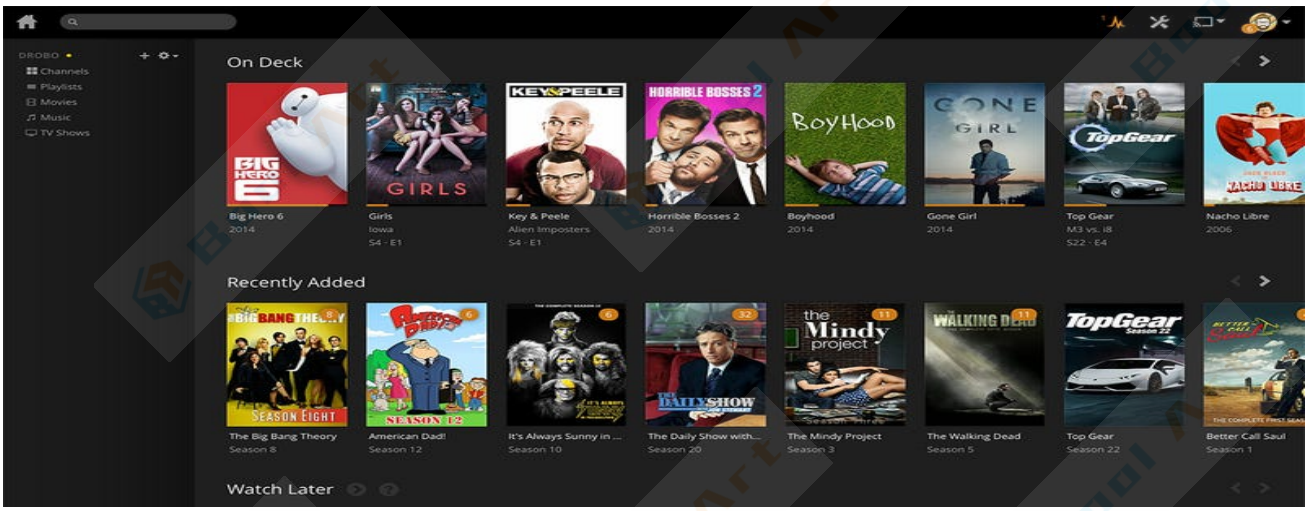
三、SVD案例（电影推荐系统）分析

数据采集的快速增长开创了信息的新时代。数据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”，因为它们可以提高搜索结果的质量，并提供与搜索项更为相关或与用户的搜索历史相对应的内容。

推荐系统用于预测用户对某项商品的评价或偏好。几乎每家大型科技公司都以某种形式应用了它们：亚马逊使用它向客户推荐产品，YouTube使用它来决定自动播放下一个要播放的视频，Facebook使用它来推荐喜欢的页面和关注的人。此外，像Netflix和Spotify这样的公司在很大程度上取决于其推荐系统对其业务和业务来源的有效性支持。

在这部分内容中，我们将通过 TMDB 5000电影数据集 (https://www.kaggle.com/tmdb/tmdb-movie-metadata) 来创建电影推荐系统的baseline。对于新手而言，这部分内容也可以作为推荐系统的入门知识。



推荐系统基本有三种类型:

- **受众特征过滤**- 根据电影的受欢迎程度和/或体裁，它们为每个用户提供通用建议。系统向具有相似受众统计特征的用户推荐相同的电影。由于每个用户都不相同，因此认为该方法太简单了。该系统背后的基本思想是，更受大众欢迎和好评的电影具有更高的被普通观众喜欢的可能性。

- **基于内容的过滤**- 也就是根据规定的那个项目，来推荐类似的项目。该系统使用项目元数据（例如电影的流派，导演，描述，演员等）来提出这些建议。这些推荐系统背后的总体思想是，如果某人喜欢某个特定项目，那么他（她）也将喜欢与之相似的项目。

- **协同过滤**- 此系统匹配具有相似兴趣的人，并根据此匹配提供推荐。协同过滤器不需要像“基于内容的推荐”那样的项目元数据。

1 加载数据集

In [1]:

```
import pandas as pd
import numpy as np
df1=pd.read_csv('lesson_3_data/tmdb_5000_movie_data/tmdb_5000_credits.csv')
df2=pd.read_csv('lesson_3_data/tmdb_5000_movie_data/tmdb_5000_movies.csv')
```

第一份数据集包含的特征:

- movie_id - 每部电影的唯一标识符。
- cast - 主角和配角的姓名。
- crew - 导演，剪辑，作曲，作家等的姓名。

第二份数据集包含的特征:

- budget - 电影制作的预算
- genre - 电影的类型：动作，喜剧，惊悚片等
- homepage - 电影首页的链接
- id - 电影的id（与第一份数据集中的一样）
- keywords - 与电影相关的关键字或标签
- original_language - 电影制作语言
- original_title - 翻译或改编之前的电影标题
- overview - 电影的简要介绍
- popularity - 表示电影的受欢迎程度的数值
- production_companies - 电影的制作公司
- production_countries - 制作国家/地区
- release_date - 上映日期
- revenue - 电影在全球范围内的票房
- runtime - 影片时长
- status - 已上映或待上映
- tagline - 电影标语
- title - 电影名称
- vote_average - 平均评分
- vote_count - 评分次数

先将两份数据集通过id列合并。

In [2]:

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

查看前5条数据

In [3]:

```
df2.head(5)
```

Out[3]:

	budget	genres	homepage	id	keywords	origina
0	237000000	[{"id": 28, "name": "Action"}, {"id": 12, "nam...	http://www.avatarmovie.com/	19995	[{"id": 1463, "name": "culture clash"}, {"id":...	
1	300000000	[{"id": 12, "name": "Adventure"}, {"id": 14, "...	http://disney.go.com/disneypictures/pirates/	285	[{"id": 270, "name": "ocean"}, {"id": 726, "na...	
2	245000000	[{"id": 28, "name": "Action"}, {"id": 12, "nam...	http://www.sonypictures.com/movies/spectre/	206647	[{"id": 470, "name": "spy"}, {"id": 818, "name...	
3	250000000	[{"id": 28, "name": "Action"}, {"id": 80, "nam...	http://www.thedarkknightrises.com/	49026	[{"id": 849, "name": "dc comics"}, {"id": 853,...	
4	260000000	[{"id": 28, "name": "Action"}, {"id": 12, "nam...	http://movies.disney.com/john-carter	49529	[{"id": 818, "name": "based on novel"}, {"id":...	

5 rows × 23 columns

2 基于受众特征过滤

开始之前：

- 我们需要一个指标来对电影进行评分
- 计算每部电影的评分
- 对分数进行排序，并向用户推荐评分最高的电影

我们可以将电影的平均评分作为得分，但只使用评分也不够合理。因为假如一部电影的平均评分有8.9，但只有3人评分；而另一部电影的平均评分只有7.8，却有40人评分，因此不能认为8.9分的电影比7.8分的更好。 因此，我将使用IMDB的加权评分（wr），该评分为：

$(\frac{v}{v+m}) \times R + (\frac{m}{m+v} \times c)$

其中,

- v 评分次数;
- m 能在排行榜中列出的最低评分次数;
- R 平均评分
- C 整个数据的平均评分

目前已经有了 v(**vote_count**) 和 R (**vote_average**) ， C 可以通过如下计算得出：

In [4]:

```
C= df2['vote_average'].mean()
C
```

Out[4]:

```
6.092171559442011
```

因此，所有电影在总分为10分的情况下，平均评分约为6。下一步是确定m的适当值，即能在排行榜中列出的最低评分次数。我们将使用第90个百分位数作为我们的分界点。换句话说，要使电影在图表中出现，其评分次数必须比列表中至少90%的电影多。

In [5]:

```
m= df2['vote_count'].quantile(0.9)
m
```

Out[5]:

```
1838.4000000000015
```

现在我们可以筛选出符合排行榜的电影

In [6]:

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

Out[6]:

```
(481, 23)
```

我们看到有481部电影符合此排行榜的资格。现在，我们需要计算每个合格电影的指标。为此，我们将定义一个函数weighted_rating()，并定义一个新特征score，通过将该函数应用于合格电影的DataFrame来计算值：

In [7]:

```
def weighted_rating(x, m=m, C=C):
    v = x['vote_count']
    R = x['vote_average']
    # 通过 IMDB 公式计算
    return (v/(v+m) * R) + (m/(m+v) * C)
```

In [8]:

```
# 定义一个新特征'score' 使用`weighted_rating()`函数计算'score'的值
q_movies['score'] = q_movies.apply(weighted_rating, axis=1)
```

最后，让我们根据'score'特征对DataFrame进行排序，并输出标题，评分次数，平均评分和score（加权评分）的前10名电影。

In [9]:

```
#根据 score 排序
q_movies = q_movies.sort_values('score', ascending=False)

#输出 top 10 的电影
q_movies[['title', 'vote_count', 'vote_average', 'score']].head(10)
```

Out[9]:

	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

至此，我们做了第一个推荐系统（尽管很基本）。在这些系统的“现在趋势”标签下，我们可以通过按“受欢迎度”列对数据集进行排序，来找到非常受欢迎的电影。

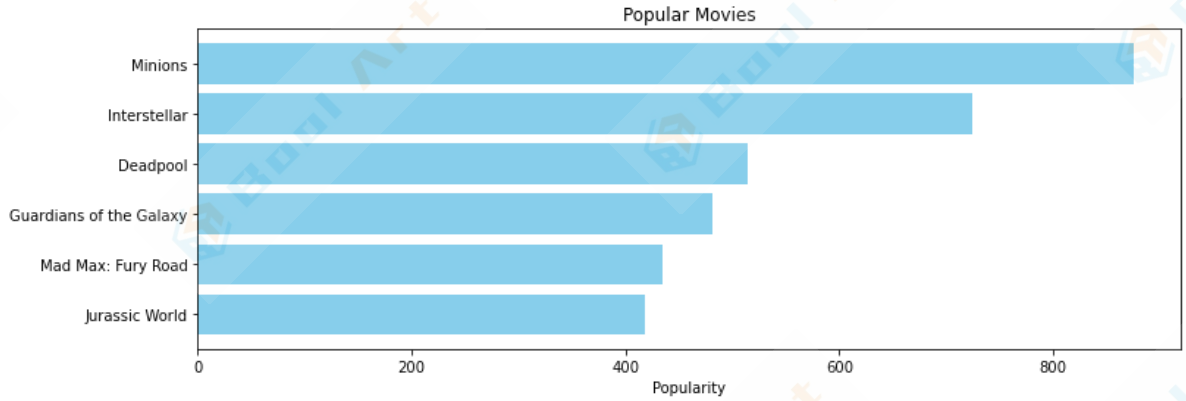
In [10]:

```
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")
```

Out[10]:

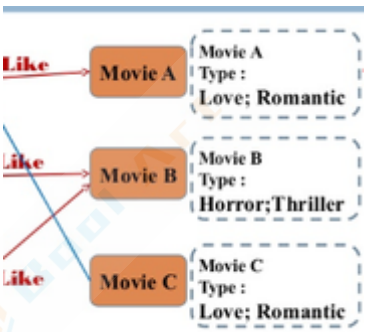
```
Text(0.5, 1.0, 'Popular Movies')
```



要知道的是，虽然基于受众特征的推荐系统向所有用户提供了推荐电影的榜单。但因为他们对特定用户的兴趣和爱好不敏感，所以我们要继续使用更完善的系统——基于内容的过滤。

3 基于内容的过滤

在此推荐器系统中，通过电影的内容（概述，演员，制作人员，关键字，标签等）查找其与其他电影的相似性。然后，推荐最可能相似的电影。



3.1 基于情节描述的推荐系统

我们将根据所有电影的情节描述计算成对相似度得分，并根据相似度得分推荐电影。“情节描述”在数据集的“overview”特征列已经给出。

In [11]:

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

Out[11]:

```
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
```

转换每个overview的词向量：为每个概述计算词频-逆文件逆频率（TF-IDF）向量。

什么是 TF-IDF?

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。

总结：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。这也就是TF-IDF的含义。

什么是TF?

TF(Term Frequency, 词频)表示词条在文本中出现的频率，这个数字通常会被归一化(一般是词频除以文章总词数), 以防止它偏向长的文件（同一个词语在长文件里可能会比短文件有更高的词频，而不管该词语重要与否）。TF用公式表示如下

$$TF_{i,j} = \frac{n_{i,j}}{\sum_k n_{k,j}}$$

其中，n_{ij}表示词条t_i在文档d_j中出现的次数，TF_{ij}就是表示词条t_i在文档d_j中出现的频率。

什么是IDF?

IDF(Inverse Document Frequency, 逆文件频率)表示关键词的普遍程度。如果包含词条i的文档越少，IDF越大，则说明该词条具有很好的类别区分能力。某一特定词语的IDF，可以由总文件数目除以包含该词语之文件的数目，再将得到的商取对数得到

$$IDF_i = \log \frac{|D|}{1 + |j : t_i \in d_j|}$$

其中，$|D|$表示所有文档的数量，$|j : t_i \in d_j|$表示包含词条t_i的文档数量，为什么这里要加1呢？主要是防止包含词条t_i的数量为0从而导致运算出错的现象发生。

某一特定文件内的高词语频率，以及该词语在整个文件集合中的低文件频率，可以产生出高权重的TF-IDF。因此，TF-IDF倾向于过滤掉常见的词语，保留重要的词语，表达为

$$TF\text{-}IDF = TF \cdot IDF$$

这会得出一个矩阵，其中的每一列代表overview中的一个单词（所有出现在至少一个文档中的单词），每一行代表一个电影，这样做是为了降低在情节描述中频繁出现的单词的重要性，也是它们在计算最终相似度得分中的重

要性。scikit-learn提供了一个内置的TfIdfVectorizer类，该类以两行代码生成TF-IDF矩阵，非常好用。

In [12]:

```
#从 scikit-learn 导入 TfIdfVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer

#定义一个TF-IDF矢量化器对象。 删除所有英语停用词，例如“ the”，“ a”
tfidf = TfidfVectorizer(stop_words='english')

#用空字符串替换NaN
df2['overview'] = df2['overview'].fillna('')

#通过拟合和转换数据来构造所需的TF-IDF矩阵
tfidf_matrix = tfidf.fit_transform(df2['overview'])

#输出tfidf_matrix的shape
tfidf_matrix.shape
```

Out[12]:

```
(4803, 20978)
```

我们看到数据集中使用了20,000多个不同的词来描述4800部电影。

有了这个矩阵，我们现在可以计算一个相似度得分。有几个候选方案，例如欧几里得，皮尔逊和[余弦相似度得分]（https://en.wikipedia.org/wiki/Cosine_similarity）。(https://en.wikipedia.org/wiki/Cosine_similarity%EF%BC%89%E3%80%82) 没有哪个分数是最好的正确答案，不同的分数在不同的情况下效果很好，尝试使用不同的指标是一个很好的分析方法。

我们将使用余弦相似度来计算表示两个电影之间相似度的数值。选择使用余弦相似度评分，是因为它与幅度无关，并且相对容易且快速地进行计算。 数学公式如下：

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

由于我们使用了TF-IDF矢量化器，因此计算点积将直接为我们提供余弦相似度评分。因此，我们将使用sklearn的linear_kernel()代替cosine_similarities()，因为它速度更快。

In [13]:

```
# 导入 linear_kernel
from sklearn.metrics.pairwise import linear_kernel

# 计算余弦相似度矩阵
cosine_sim = linear_kernel(tfidf_matrix, tfidf_matrix)
```

我们将定义一个函数，该函数以电影标题作为输入并输出10个最相似电影的列表。为此，我们需要电影标题和DataFrame索引的反向映射。换句话说，我们需要一种机制来确定电影在我们的元数据DataFrame中的索引。

In [14]:

```
#构造索引和电影标题的反向映射
indices = pd.Series(df2.index, index=df2['title']).drop_duplicates()
```

现在，就是定义推荐系统的好时候，以下是我们将遵循的步骤：

- 根据给定的标题获取电影的索引。
- 获取该特定电影与所有电影的余弦相似度得分列表。将其转换为元组列表，其中第一个元素是其位置，第二个元素是相似度得分。
- 根据相似度得分对上述元组列表进行排序；也就是第二个元素。
- 获取此列表的前10个元素。忽略第一个涉及电影本身的元素（与特定电影最相似的电影是电影本身）。
- 返回与顶部元素的索引相对应的标题。

In [15]:

```
# 定义输入电影名称，输出与之最相似的电影的函数
def get_recommendations(title, cosine_sim=cosine_sim):
    # 获取与名称匹配的电影的索引
    idx = indices[title]

    # 获取所有电影与该电影的相似度得分
    sim_scores = list(enumerate(cosine_sim[idx]))

    # 根据相似度得分对电影进行排序
    sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)

    # 获取10部最相似的电影的相似度得分
    sim_scores = sim_scores[1:11]

    # 获取电影索引
    movie_indices = [i[0] for i in sim_scores]

    # 返回10部最相似的电影
    return df2['title'].iloc[movie_indices]
```

In [16]:

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

Out[16]:

```
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
```

In [17]:

```
get_recommendations('The Avengers')
```

Out[17]:

```
7               Avengers: Age of Ultron
3144                             Plastic
1715                             Timecop
4124                 This Thing of Ours
3311               Thank You for Smoking
3033                       The Corruptor
588     Wall Street: Money Never Sleeps
2136          Team America: World Police
1468                        The Fountain
1286                         Snowpiercer
Name: title, dtype: object
```

尽管我们做的系统在查找具有类似剧情描述的电影方面做得不错，但是推荐的质量却不是那么好。“The Dark Knight Rises”将重现所有蝙蝠侠电影，而喜欢该电影的人们更有可能更喜欢 Christopher Nolan 的其他电影，这是本系统无法捕获的。

3.2 基于品质, 类型 和关键字的推荐系统

在本节中要做的是使用更好的元数据，提高我们推荐系统的质量。我们将基于以下元数据构建推荐器：3个主要演员，导演，电影类型和电影情节关键字。

从演员，剧组和关键字特征中，我们需要提取与该电影相关的三个最重要的演员，导演和关键字。我们现在的数据是“字符串化”列表的形式，需要将其转换为安全且可用的结构。

In [18]:

```
# 将字符串化的特征解析为它们对应的python对象
from ast import literal_eval

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

接下来，我们将定义从每个特征中提取所需信息的函数。

In [19]:

```
# 从剧组特征中获取导演的姓名，如果没有列出director，则返回NaN
def get_director(x):
    for i in x:
        if i['job'] == 'Director':
            return i['name']
    return np.nan
```

In [20]:

```
# 返回列表的前3个元素或整个列表
def get_list(x):
    if isinstance(x, list):
        names = [i['name'] for i in x]
        #检查是否存在3个以上的元素，如果是，则仅返回前三个；如果否，则返回整个列表。
        if len(names) > 3:
            names = names[:3]
        return names

    #如果数据丢失/格式错误，则返回空列表
    return []
```

In [21]:

```
# 以合适的形式定义新的导演，演员，类型和关键字特征。
df2['director'] = df2['crew'].apply(get_director)

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

In [22]:

```
# 输出前三部电影的新特征
df2[['title', 'cast', 'director', 'keywords', 'genres']].head(3)
```

Out[22]:

	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]

下一步是将电影名称和关键字实例转换为小写并去除它们之间的所有空格。这样做是为了使我们的矢量化程序不会将“ Johnny Depp”和“ Johnny Galecki”的Johnny认作相同的。

In [23]:

```
# 将所有字符串转换为小写并去除空格的函数
def clean_data(x):
    if isinstance(x, list):
        return [str.lower(i.replace(" ", "")) for i in x]
    else:
        #检查目录是否存在。如果不存在，则返回空字符串。
        if isinstance(x, str):
            return str.lower(x.replace(" ", ""))
        else:
            return ''
```

In [24]:

```
# 对特征应用clean_data函数
features = ['cast', 'keywords', 'director', 'genres']

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

现在，就可以创建“metadata soup”，它是一个字符串，包含我们要提供给矢量化器的所有元数据（即演员，导演和关键字）。

In [25]:

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

后续步骤与我们对基于情节描述的推荐器所做的相同。一个重要的区别是我们使用CountCountizer()而不是TF-IDF，这是因为我们不希望减轻演员/导演在相对较多的电影中的表演或导演的影响。

In [26]:

```
# 导入CountVectorizer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'])
```

In [27]:

```
# 根据count_matrix计算余弦相似度矩阵
from sklearn.metrics.pairwise import cosine_similarity

cosine_sim2 = cosine_similarity(count_matrix, count_matrix)
```

In [28]:

```
# 重置主DataFrame的索引，像前面一样构造反向映射
df2 = df2.reset_index()
indices = pd.Series(df2.index, index=df2['title'])
```

现在，我们可以通过传入新的**cosine_sim2**矩阵作为第二个参数来重用 **get_recommendations()** 函数。

In [29]:

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

Out[29]:

```
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
```

In [30]:

```
get_recommendations('The Godfather', cosine_sim2)
```

Out[30]:

```
867      The Godfather: Part III
2731      The Godfather: Part II
4638    Amidst the Devil's Wings
2649           The Son of No One
1525              Apocalypse Now
1018             The Cotton Club
1170     The Talented Mr. Ripley
1209               The Rainmaker
1394               Donnie Brasco
1850                    Scarface
Name: title, dtype: object
```

可以看到，由于更多的元数据，我们的推荐系统已成功捕获了更多信息，并为我们提供了更好的推荐。Marvels or DC comics粉丝更有可能会喜欢同一家制作公司的电影。因此，在上述特征中，我们可以添加进*production_company*。 通过在soup中多次添加特征，还可以增加Director的权重。

4 协同过滤

我们基于内容的过滤受到一些严重的限制。它仅能推荐接近某个电影的电影，也就是说，它无法捕捉口味并提供跨类型的推荐。

同时，我们构建的推荐系统不够真正的个性化，因为它无法捕获用户的个人品味和偏见。任何以电影为基础使用我们的推荐系统的人，都会获得与该电影相同的推荐。

所以接下来，将使用一种称为“协同过滤”的技术来向“电影观看者”提出建议。

它基本上有两种类型：

- **基于用户的过滤**- 系统向用户的相似用户推荐用户喜欢的产品。为了测量两个用户之间的相似性，我们可以使用皮尔逊相关性或余弦相似性。该过滤方法可以用一个例子来说明，在以下矩阵中，每一行代表一个用户，而各列对应于不同的电影，但最后一个电影记录了该用户与目标用户之间的相似度。每个单元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

由于用户A和F没有与用户E有共同的电影评分，它们与用户E的相似性在Pearson Correlation中没有定义。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

从上表中可以看出，用户D与用户E有很大不同，因为它们之间的Pearson相关系数为负。他给《Me Before You》的评分高于他的平均评分，而用户E则相反。现在，我们可以开始填写用户E尚未根据其他用户评分的电影的表格。

	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

尽管user-based CF计算非常简单，但是它仍存在一些问题。一个主要问题是用户的喜好会随着时间而改变，这就表明基于相邻用户的预计算矩阵可能会导致性比较差的表现。为了解决这个问题，我们可以应用item-based CF。

- **基于内容的协同过滤**- item-based CF不会根据用户之间的相似度进行推荐，而是根据其与目标用户所评价内容的相似性来推荐内容。同样，可以使用皮尔逊相关性或余弦相似性来计算相似度。主要区别在于，与user-based CF的水平填写方式相反，对于item-based CF的协作过滤，我们竖着填写表格。下表示例如何给电影《Me Before You》完成这个内容。

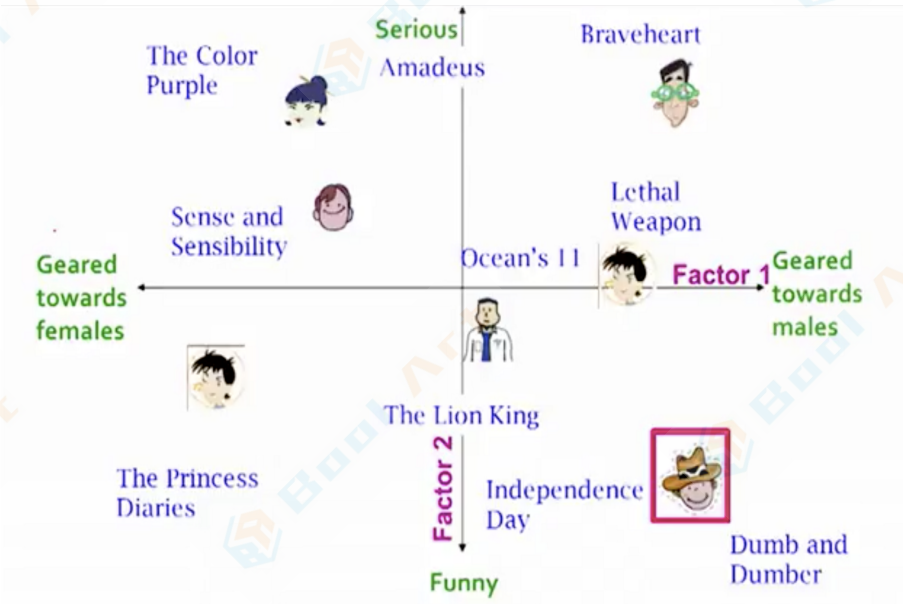
	The Avengers	Sherlock	Transformers	Matrix	Titanic	Me Before You
A	2		2	4	5	2.94*
B	5		4			1
C			5		2	2.48*
D		1		5		4
E			4			2
F	4	5		1		1.12*
Similarity	-1	-1	0.86	1	1	

由于item-based CF更静态，因此它成功避免了动态用户喜好带来的问题。但是，这种方法仍然存在一些问题。首先，主要问题是***可扩展性***。 计算量随着用户和内容一起增长。最坏的情况是复杂度为O（mn），其中m个用户n个内容。***稀疏性***是另一个问题。再次查看上表，尽管只有一个用户同时评价了Matrix和Titanic，但它们之间的相似度已经达到了1。在极端情况下，我们可以拥有数百万个用户，而两部完全不同的电影之间的相似度可能很高，可能就是因为一个用户给两者的评分相似。

4.1 单值分解

处理CF创建的可伸缩性和稀疏性问题的一种方法是利用“潜在因子模型”来捕获用户和项目之间的相似性。本质上，我们希望将推荐问题转化为优化问题，可以将其视为预测给定用户物品评分的准确度。一种常见的度量方式是均方根误差（RMSE）。**RMSE越低，性能越好**。

现在谈论”潜在因子“时，你可能想知道这是什么？这是一个广义的概念，它描述了用户或物品具有的属性或概念。例如，对于音乐，潜在因子可以指音乐所属的流派。SVD通过提取其潜在因子来减小效用矩阵的维数。本质上，我们将每个用户和每个项目映射到维度为r的潜在空间中。因此，当它们直接可比时，它可以帮助我们更好地了解用户与物品之间的关系。下图说明了这种想法。



说了这么多，让我们看看要如何实现这一点。 由于之前使用的数据集没有userId（这是协作过滤所必需的），因此需要加载另一个数据集。我们将使用**Surprise** (https://surprise.readthedocs.io/en/stable/index.html)来实现SVD。

In [31]:

```
from surprise import Reader, Dataset, SVD
from surprise.model_selection import cross_validate
from sklearn.model_selection import RepeatedKFold
reader = Reader()
ratings = pd.read_csv('lesson_3_data/movie_dataset/ratings_small.csv')
ratings.head()
```

Out[31]:

	userId	movieId	rating	timestamp
0	1	31	2.5	1260759144
1	1	1029	3.0	1260759179
2	1	1061	3.0	1260759182
3	1	1129	2.0	1260759185
4	1	1172	4.0	1260759205

需要注意，在这份数据集中，电影的评分为5分制，与之前的评分制不同。

In [32]:

```
from sklearn.model_selection import KFold
data = Dataset.load_from_df(ratings[['userId', 'movieId', 'rating']], reader)
```

In [33]:

```
svd = SVD()
cross_validate(svd, data, measures=['RMSE', 'MAE'], cv=5, verbose=True)
```

```
Evaluating RMSE, MAE of algorithm SVD on 5 split(s).

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    0.9130  0.8992  0.8922  0.8883  0.8912  0.8968  0.00
89
MAE (testset)     0.7007  0.6929  0.6885  0.6832  0.6871  0.6905  0.00
60
Fit time          3.79    3.84    3.78    3.86    3.87    3.83    0.04
Test time         0.09    0.13    0.08    0.08    0.08    0.09    0.02
```

Out[33]:

```
{'test_rmse': array([0.91296411, 0.89923454, 0.89216406, 0.88826117,
0.89116527]),
 'test_mae': array([0.70071517, 0.69292427, 0.68845076, 0.68316958, 0.
68712697]),
 'fit_time': (3.794041156768799,
  3.840475082397461,
  3.7773020267486572,
  3.8602330684661865,
  3.870151996612549),
 'test_time': (0.08837890625,
  0.1326448917388916,
  0.08184671401977539,
  0.08222699165344238,
  0.08197999000549316)}
```

我们得到的平均均方根误差约为0.89，对于我们的情况而言已经可以了。接下来对数据集进行训练并得出预测。

In [34]:

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

Out[34]:

```
<surprise.prediction_algorithms.matrix_factorization.SVD at 0x7f8ed1ba
fdc0>
```

先选择ID为1的用户，并检查她/他给出的评分。

In [35]:

```
ratings[ratings['userId'] == 1]
```

Out[35]:

	userId	movieId	rating	timestamp
0	1	31	2.5	1260759144
1	1	1029	3.0	1260759179
2	1	1061	3.0	1260759182
3	1	1129	2.0	1260759185
4	1	1172	4.0	1260759205
5	1	1263	2.0	1260759151
6	1	1287	2.0	1260759187
7	1	1293	2.0	1260759148
8	1	1339	3.5	1260759125
9	1	1343	2.0	1260759131
10	1	1371	2.5	1260759135
11	1	1405	1.0	1260759203
12	1	1953	4.0	1260759191
13	1	2105	4.0	1260759139
14	1	2150	3.0	1260759194
15	1	2193	2.0	1260759198
16	1	2294	2.0	1260759108
17	1	2455	2.5	1260759113
18	1	2968	1.0	1260759200
19	1	3671	3.0	1260759117

In [36]:

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

Out[36]:

```
Prediction(uid=1, iid=302, r_ui=3, est=2.586449525024487, details={'wa
s_impossible': False})
```

对于影片ID为302的电影，我们获得的预测值为**2.586**。此推荐器系统的一个令人震惊的功能是它不在乎电影是什么（电影内容包含哪些）。它纯粹基于分配的影片ID进行工作，并尝试根据其他用户对该影片的预测来预测评分。

总结

我们使用受众特征过滤，基于内容的过滤和协同过滤来创建推荐系统。虽然受众特征过滤非常简单，但却无法实际使用。**混合系统**可以利用基于内容的过滤和协同过滤，因为事实证明这两种方法几乎是互补的。该模型是非常基础的参照模型，仅提供了一个入门的框架。

参考资料

- https://hackernoon.com/introduction-to-recommender-system-part-1-collaborative-filtering-singular-value-decomposition-44c9659c5e75 (https://hackernoon.com/introduction-to-recommender-system-part-1-collaborative-filtering-singular-value-decomposition-44c9659c5e75)
- https://www.kaggle.com/rounakbanik/movie-recommender-systems (https://www.kaggle.com/rounakbanik/movie-recommender-systems)
- http://trouvus.com/wp-content/uploads/2016/03/A-hybrid-movie-recommender-system-based-on-neural-networks.pdf (http://trouvus.com/wp-content/uploads/2016/03/A-hybrid-movie-recommender-system-based-on-neural-networks.pdf)