案例三：使用Python基于K近邻实现商品推荐

1 案例目的

• 掌握K近邻的使用场景；

• 理解KNN（K近邻）的原理；

• 掌握使用K近邻实现推荐类的过程。

2 案例内容

使用K近邻算法实现电影的推荐，即根据你喜欢的影片推荐与其相似的影片。

3 案例知识点

• K近邻算法基本原理；

• 使用K近邻算法实现电影推荐。

4 案例时长

共2学时，具体安排如下：

• 处理数据（1.4学时）

• 训练模型（0.3学时）

• 预测结果（0.3学时）

5 案例实验环境

**•操作系统：**

1）Linux Ubuntu 16.04 64位操作系统

**•软件环境：**

1）Python 3.7

2）sklearn 0.23

**•开发环境与工具：**

1）Spyder 4

6 案例分析

本案例使用的是无监督KNN。

处理数据，使得电影条数为行，打分用户数为列（如电影数量是1000部，打分用户数是800个，那么数据就是1000行800列）。

使用无监督KNN基于余弦相似度找到距离较近的电影，从而实现电影推荐。

7 案例实验过程

使用K近邻分类算法实现电影的推荐类，可分为以下3个步骤：

1、处理数据；

2、训练模型；

3、预测结果。

7.1准备工作

点击【桌面连接】进入案例实验环境。

在桌面上新建一个目录，名字叫movie-recommendation。本案例中所有的数据集和代码都放在该目录下。

7.1.1 获取数据

在刚创建的movie-recommendation目录下新建一个目录，命名为data。用于存储本案例用到的数据集。

本案例需要2个数据集，分别为movies.csv和ratings.csv。

点击右侧菜单栏中的【资源下载】按钮，复制提供的URL前缀，如https://xxx.eec-cn.com/。

将URL前缀和数据集地址拼接到一起，如https://xxx.eec-cn.com/dataSet/systemLib/d0125b3c401f4ea7a925ae475c1dcb62.zip。下载数据集后解压，会解压出2个csv文件，分别为movies.csv和ratings.csv。将这2个csv文件复制到刚创建的movie-recommendation/data目录下。

7.1.2 打开Spyder软件

在桌面上点击Spyder图标打开Spyder，并把工作目录切换至movie-recommendation目录下。

7.1.3 新建代码文件

确保Spyder的工作目录为movie-recommendation。使用Spyder新建一个Python文件，命名为movie\_recommendation.py。

至此，整个案例的目录结构如图1所示。

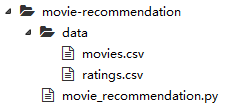


图1 案例的目录结构

7.1.4 了解数据

movies.csv数据集的结构如图2所示。其中，movieId表示电影序号；title表示电影名称。

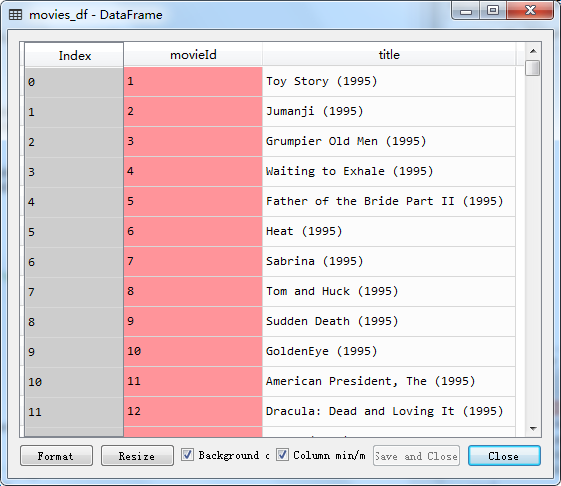


图2 movies.csv数据集结构

ratings.csv数据集的结构如图3所示。其中，userId表示参与评分的用户ID；movieId表示电影序号，和movies.csv中的movieId一一对应；rating表示用户为该电影打的分，分数在0~5之间。

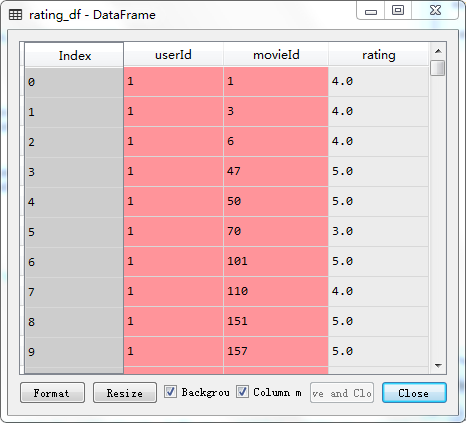


图3 ratings.csv数据集结构

7.2处理数据

处理数据，使得电影条数为行，打分用户数为列。

7.2.1 导入包

【代码7-2-1】movie\_recommendation.py

# -\*- coding: utf-8 -\*-

# 导入包

import numpy as np

import pandas as pd

from scipy.sparse import csr\_matrix

from sklearn.neighbors import NearestNeighbors # 无监督KNN

# from sklearn.neighbors import KNeighborsClassifier

执行代码块。

7.2.2 导入数据集

数据集有2个，分别是电影数据和评分数据。

【代码7-2-2】movie\_recommendation.py

# 导入数据集

movies\_df = pd.read\_csv('data/movies.csv',usecols=['movieId','title'],

dtype={'movieId': 'int32', 'title': 'str'})

ratings\_df = pd.read\_csv('data/ratings.csv',usecols=['userId', 'movieId', 'rating'],

dtype={'userId': 'int32', 'movieId': 'int32', 'rating': 'float32'})

执行代码块。

此时建议切换到Spyder的Variable explorer面板，便于查看变量信息。

movies\_df变量的值如图4所示。

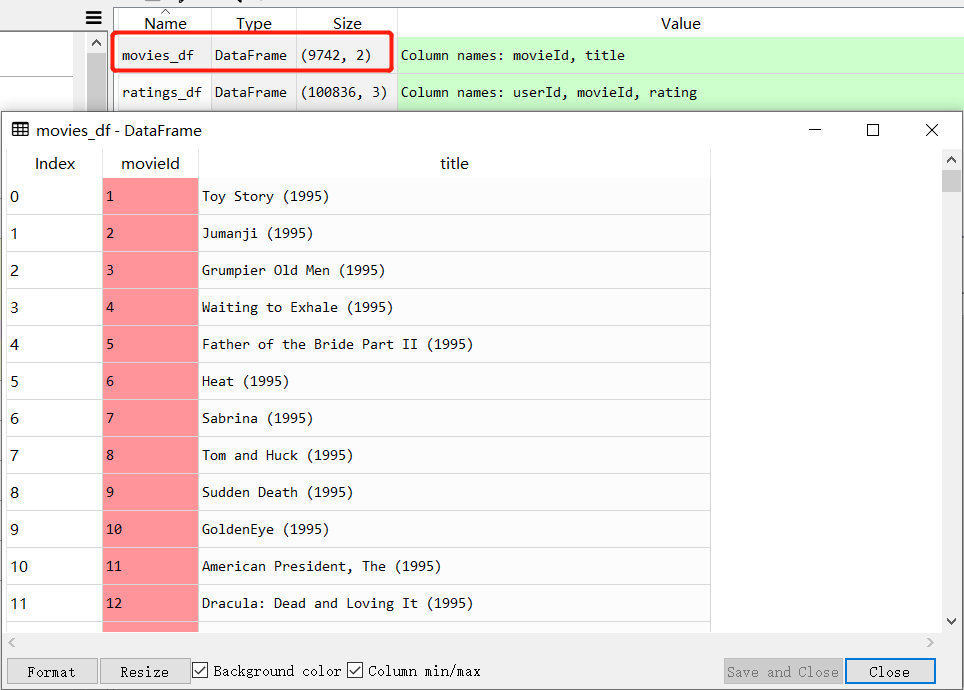


图4 movies\_df变量的值

ratings\_df变量的值如图5所示。

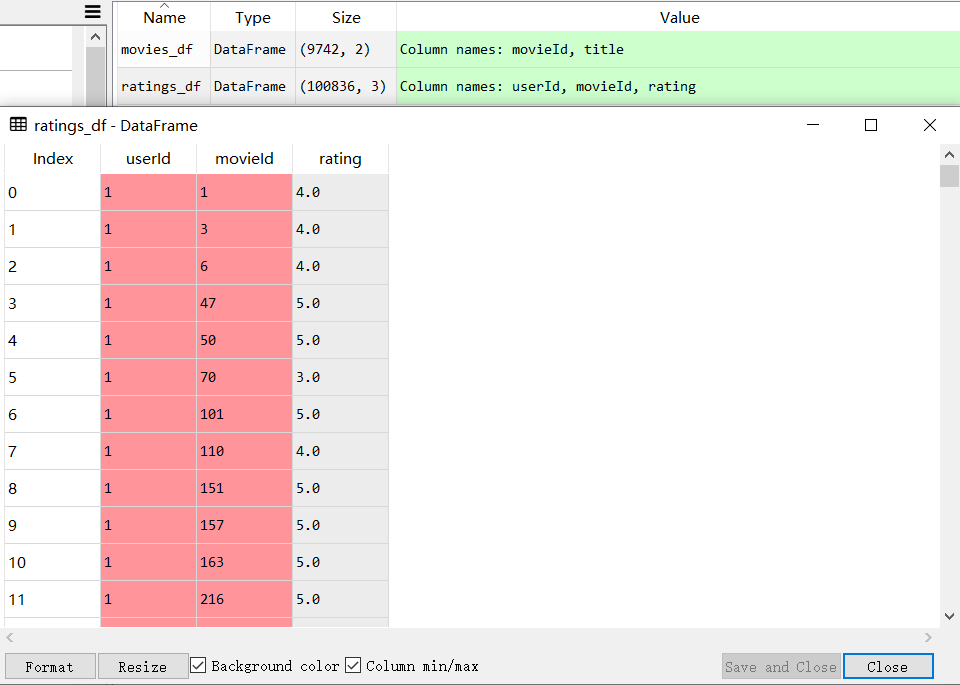


图5 ratings\_df变量的值

7.2.3 合并数据集

将数据集合并做分析。

【代码7-2-3】movie\_recommendation.py

movies\_df和ratings\_df可以通过movieId字段合并到一起。

# 合并数据集

df = pd.merge(ratings\_df,movies\_df,on='movieId')

执行代码块。

合并之后，movies\_df和ratings\_df的内容存储到df变量中。df变量的值如图6所示。

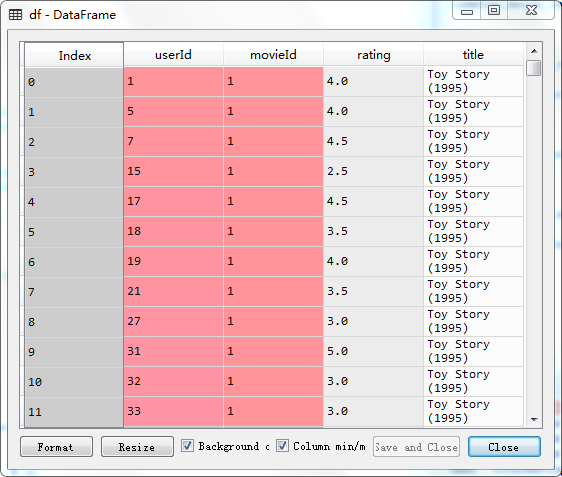


图6 df变量的值

7.2.4 检查数据集是否有异常

在处理数据集之前，首先应检查数据集是否有异常值、缺失值等。

【代码7-2-4】movie\_recommendation.py

# 检查数据集是否有异常

df\_describe = df.describe()

df\_null = df.isnull().sum()

执行代码块。

df\_describe变量的值如图7所示。

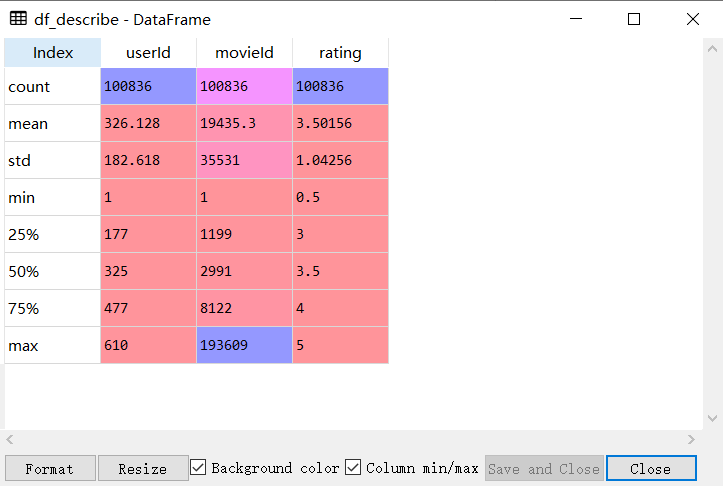


图7 df\_describe变量的值

由上图可见，rating列最小值是0.5，最大值是5，范围合理，没有异常。

df\_null变量的值如图8所示。



图8 df\_null变量的值

由上图可见，所有列都没有缺失值。

7.2.5 去掉评分人数少的电影

有的电影评分人数较少，这样的数据不利于构建模型。应该删除评分人数较少的电影。删除之前首先统计每部电影的评分人数。

【代码7-2-5-1】movie\_recommendation.py

# 统计每部电影的评分人数

movie\_rating\_count\_df = df.groupby(by = ['title'])['rating'].count().reset\_index()

movie\_rating\_count\_df.columns=['title', 'total\_rating\_count']

total\_rating\_count\_desc = movie\_rating\_count\_df.describe()

执行代码块。

每部电影的评分人数详细信息存储在movie\_rating\_count\_df变量中。每部电影的评分人数统计信息存储在total\_rating\_count\_desc变量中。total\_rating\_count\_desc变量的值如图9所示。

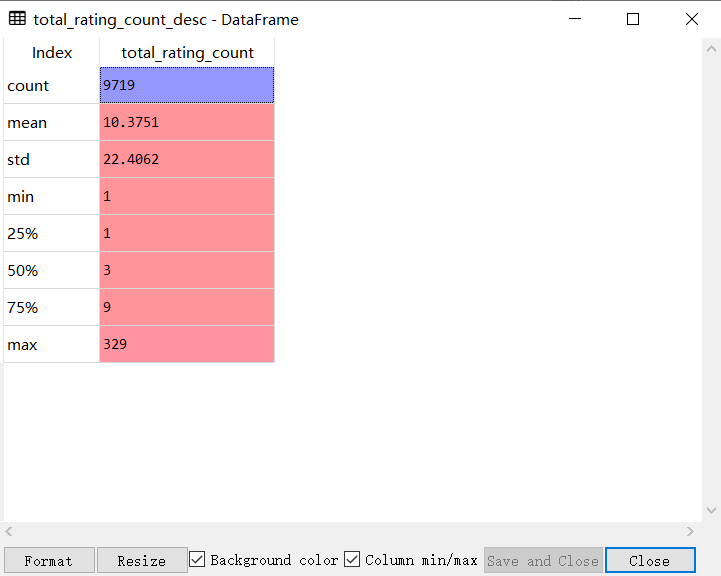


图9 total\_rating\_count\_desc变量的值

由上图可见，在9719部电影中，电影的评分人数最少是1人，最多是329人。

下面删除电影评分人数少的电影。

首先将原始数据集与电影评分人数数据集合并，然后根据评分人数筛选数据集。本案例认为评分人数少于50人为不合格的数据，应该删除。

【代码7-2-5-2】movie\_recommendation.py

# 去掉评分人数少的电影

dataset = pd.merge(df, movie\_rating\_count\_df, on = 'title', how = 'left')

popularity\_threshold = 50 # 电影评分人数最小值

rating\_popular\_movie\_df = dataset.query('total\_rating\_count >= @popularity\_threshold')

执行代码块。

rating\_popular\_movie\_df变量存储着数据集完整的信息，且这些数据都是我们需要的。rating\_popular\_movie\_df变量的值如图10所示。

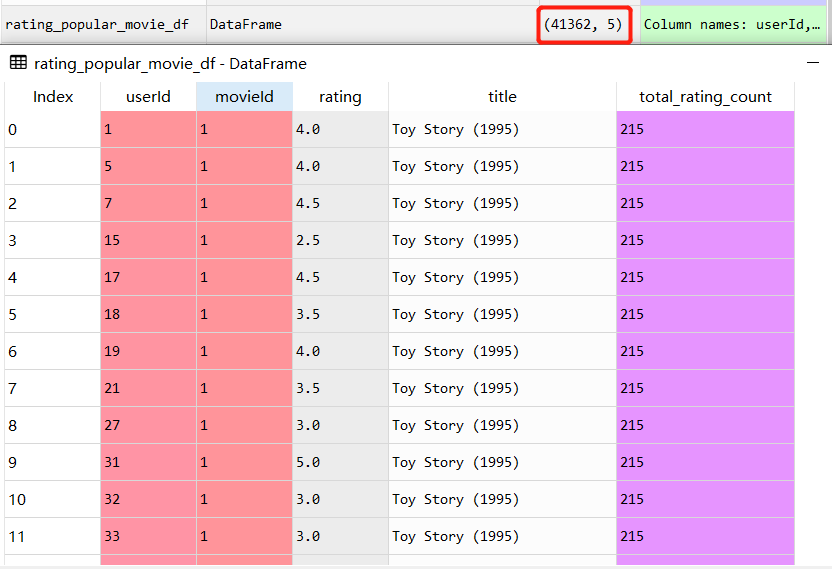


图10 rating\_popular\_movie\_df变量的值

删除评分人数少的电影后，统计一下数据集中剩余的数据有几部电影，几个用户评分。

【代码7-2-5-3】movie\_recommendation.py

# 统计筛选后的数据评论人数和电影数目

user\_count = rating\_popular\_movie\_df['userId'].unique()

print('user\_count:', user\_count.shape[0])

movie\_count = rating\_popular\_movie\_df['title'].unique()

print('movie\_count:', movie\_count.shape[0])

执行代码块。

数据集中剩余数据的电影数和评分用户数的结果如图11所示。



图11 数据集中剩余数据的电影数和评分用户数的结果

7.2.6 处理成模型需要的数据格式

数据集中剩余数据的电影数是450，评分用户数是606。模型需要的数据是一个450行606列的二维数组。数组中每个元素的值是用户的打分。如果用户没有打分则是0分。

【代码7-2-6-1】movie\_recommendation.py

# 处理成模型需要的数据格式

movie\_features\_df=rating\_popular\_movie\_df.pivot\_table(index='title',columns='userId',values='rating').fillna(0)

执行代码块。

movie\_features\_df变量的值如图12所示。

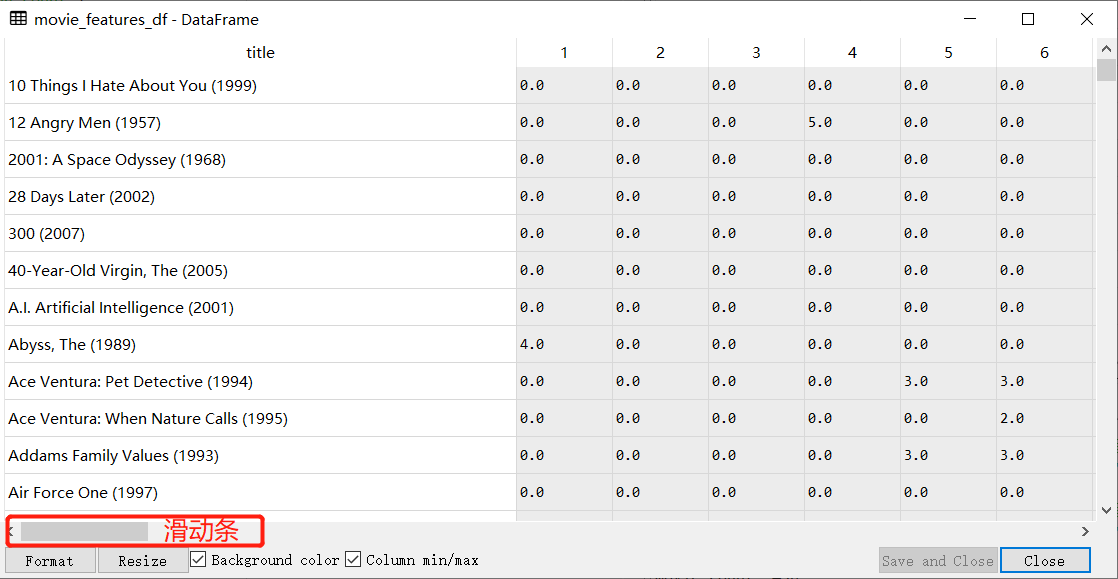


图12 movie\_features\_df变量的值

导入的KNN的类NearestNeighbors需要传入到数据格式是稀疏矩阵类型。所以需要把movie\_features\_df转换成稀疏矩阵类型。

【代码7-2-6-2】movie\_recommendation.py

# 将数据转换为压缩稀疏行矩阵

movie\_features\_df\_matrix = csr\_matrix(movie\_features\_df.values)

执行代码块。

数据准备完毕，下一步需要使用K近邻算法训练模型。

7.3训练模型

训练模型相对简单，直接调用sklearn库提供的类即可。

【代码7-3】movie\_recommendation.py

# 训练模型

model\_knn = NearestNeighbors(metric = 'cosine', algorithm = 'brute')

model\_knn.fit(movie\_features\_df\_matrix)

执行代码块。

algorithm参数设置的值是brute，意味着暴力搜索。brute是常用的参数值。

metric参数设置的值是cosine。metric的值设为cosine而不设为欧几里得距离，原因如下：

根据欧氏距离和余弦相似度各自的计算方式和衡量特征，分别适用于不同的数据分析模型：欧氏距离能够体现个体数值特征的绝对差异，所以更多的用于需要从维度的数值大小中体现差异的分析，如使用用户行为指标分析用户价值的相似度或差异；而余弦相似度更多的是从方向上区分差异，而对绝对的数值不敏感，更多的用于使用用户对内容评分来区分用户兴趣的相似度和差异，同时修正了用户间可能存在的度量标准不统一的问题（因为余弦相似度对绝对数值不敏感）。

7.4预测结果

从数据集中随机抽取一部电影。

【代码7-4-1】movie\_recommendation.py

# 随机抽取一部电影

query\_index = np.random.choice(movie\_features\_df.shape[0])

print('随机抽取的电影是：', movie\_features\_df.index[query\_index])

执行代码块。

程序执行结果如图13所示。



图13 随机抽取的电影

接下来找到所抽取的电影的最近邻居对应的距离和索引。

【代码7-4-2】movie\_recommendation.py

# 找到所抽取的电影的最近邻居对应的距离和索引

distances, indices = model\_knn.kneighbors(movie\_features\_df.iloc[query\_index,:].values.reshape(1, -1), n\_neighbors = 6)

执行代码块。

n\_neighbors参数的值设为6。这个数字6包含本身+5个邻居（训练模型时使用了默认参数：5）。5个邻居加本身正好是6个。

最后输出预测结果。

【代码7-4-3】movie\_recommendation.py

# 输出推荐结果

for i in range(0, len(distances.flatten())):

if i == 0: # 该电影本身

continue

else:

recommeded\_movie\_name = movie\_features\_df.index[indices.flatten()[i]]

movie\_distance = distances.flatten()[i]

print('推荐的第{0}部电影是: {1}, (distance: {2}:)'.format(i, recommeded\_movie\_name, movie\_distance))

执行代码块。

推荐的另外5部电影如图14所示。

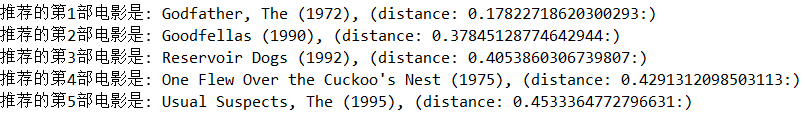


图14 推荐的另外5部电影

由上图可见，我们对《教父（1974）》这部电影进行推荐，推荐了《教父（1972）》、《教父（1990）》、《落水狗（1992）》、《飞越疯人院（1975）》、《非常嫌疑犯（1995）》。通过某一部《教父》推荐了另外两部《教父》，准确率还不错。

8 案例代码

【案例代码】movie\_recommendation.py

# -\*- coding: utf-8 -\*-

# 导入包

import numpy as np

import pandas as pd

from scipy.sparse import csr\_matrix

from sklearn.neighbors import NearestNeighbors

# 导入数据集

movies\_df = pd.read\_csv('data/movies.csv',usecols=['movieId','title'],

dtype={'movieId': 'int32', 'title': 'str'})

ratings\_df = pd.read\_csv('data/ratings.csv',usecols=['userId', 'movieId', 'rating'],

dtype={'userId': 'int32', 'movieId': 'int32', 'rating': 'float32'})

# 合并数据集

df = pd.merge(ratings\_df,movies\_df,on='movieId')

# 检查数据集是否有异常

df\_describe = df.describe()

df\_null = df.isnull().sum()

# 统计每部电影的评分人数

movie\_rating\_count\_df = df.groupby(by = ['title'])['rating'].count().reset\_index()

movie\_rating\_count\_df.columns=['title', 'total\_rating\_count']

total\_rating\_count\_desc = movie\_rating\_count\_df.describe()

# 去掉评分人数少的电影

dataset = pd.merge(df, movie\_rating\_count\_df, on = 'title', how = 'left')

popularity\_threshold = 50 # 电影评分人数最小值

rating\_popular\_movie\_df = dataset.query('total\_rating\_count >= @popularity\_threshold')

# 统计筛选后的数据评论人数和电影数目

user\_count = rating\_popular\_movie\_df['userId'].unique()

print('user\_count:', user\_count.shape[0])

movie\_count = rating\_popular\_movie\_df['title'].unique()

print('movie\_count:', movie\_count.shape[0])

# 处理成模型需要的数据格式

movie\_features\_df=rating\_popular\_movie\_df.pivot\_table(index='title',columns='userId',values='rating').fillna(0)

# 将数据转换为压缩稀疏行矩阵

movie\_features\_df\_matrix = csr\_matrix(movie\_features\_df.values)

# 训练模型

model\_knn = NearestNeighbors(metric = 'cosine', algorithm = 'brute')

model\_knn.fit(movie\_features\_df\_matrix)

# 随机抽取一部电影

query\_index = np.random.choice(movie\_features\_df.shape[0])

print('随机抽取的电影是：', movie\_features\_df.index[query\_index])

# 找到所抽取的电影的最近邻居对应的距离和索引

distances, indices = model\_knn.kneighbors(movie\_features\_df.iloc[query\_index,:].values.reshape(1, -1), n\_neighbors = 6)

# 输出推荐结果

for i in range(0, len(distances.flatten())):

if i == 0: # 该电影本身

continue

else:

recommeded\_movie\_name = movie\_features\_df.index[indices.flatten()[i]]

movie\_distance = distances.flatten()[i]

print('推荐的第{0}部电影是: {1}, (distance: {2}:)'.format(i, recommeded\_movie\_name, movie\_distance))