第5章

影评数据分析与电影推荐


视频讲解

数据分析是信息时代的一个基础而又重要的工作。面对飞速增长的数据，如何从这些数据中挖掘到更有价值的信息成为一个重要的研究方向。机器学习在各个领域的应用逐渐成熟，它已成为数据分析和人工智能的重要工具。数据分析和挖掘的一个很重要的应用领域是推荐服务。本章将利用机器学习进行影评数据分析，并结合分析结果向用户推荐电影，从而展示数据分析的整个过程。一般来说，数据分析可以简单划分为明确分析目标，数据采集、清洗和整理，数据建模和分析，以及结果展示或服务部署。在本章的实践中，这些步骤都会有所体现。

5.1 明确目标与准备数据

本案例的最终目标比较明确：根据用户对不同电影的评分情况来推荐新的电影。而要实现这个目标，其阶段性的目标可能包含找出和某用户有类似观影爱好的用户、找出和某一个电影有相似的观众群的电影等。要实现这些阶段性目标，首先要准备分析所需的数据。

在进行数据采集时，需要根据实际的业务环境采用不同的方式，如使用爬虫、对接数据库、使用接口等。有时在进行监督学习时，需要对采集的数据进行手动标记。由于本案例需要的是用户对电影的评分数据，因此可以使用爬虫获取豆瓣电影影评数据。需要注意的是，与用户信息相关的数据需要进行脱敏处理。因为本案例使用的是开源的数据，而且爬虫不是本章的重点，所以在此不再进行说明。

获取的数据有两个文件：包含加密的用户 ID、电影 ID、评分值的用户评分文件 ratings.csv 和包含电影 ID、电影名称的电影信息文件 movies.csv。因为数据比较简单，所以基本上可以省去特征方面的复杂处理过程。在实际操作中，如果无法保证获取的数据质量，就需要对数据进行清洗，包括对数据格式的统一、缺失数据的补充等。在数据清

洗完成后，还需要对数据进行整理，如根据业务逻辑进行分类、去除冗余数据等。在数据整理完成之后需要选择合适的特征，特征的选择也会根据后续的分析进行变化。关于特征的处理有一个专门的研究方向——特征工程，它也是数据分析过程中很重要且耗时的部分。

5.2　工具选择

在实现目标之前，需要对数据进行统计分析，从而了解数据的分布情况，以及数据的质量是否能够支撑我们的目标。Pandas 很适合完成这项工作。

开发工具选择比较适合尝试性开发的 Jupyter Notebook。Jupyter Notebook 是一个交互式笔记本，支持运行 40 多种编程语言。它的本质是一个 Web 应用程序，便于创建和共享文学化程序文档，支持实时代码、数学方程、可视化和 markdown。由于其灵活交互的优势，因此很适合探索性质的开发工作。其安装和使用都比较简单，这里不再做详细介绍。使用方式推荐 VS Code 开发工具，它可以直接支持 Jupyter Notebook，不需要手动启动服务，界面如图 5-1 所示。

图 5-1　VS Code Jupyter Notebook 界面展示

5.3　初步分析

准备好环境和数据之后，需要对数据进行初步的分析，一方面可以初步了解数据的构成；另一方面可以判断数据的质量。数据初步分析往往是统计性的、多角度的，带有很大的尝试性。然后再根据得到的结果进行深入的挖掘，得到更有价值的结果。对于当前的数据，可以分别从用户和电影两个角度入手。

在进入初步分析之前，需要先导入基础的用户评分数据和电影信息数据，代码如下所示。

```
import pandas as pd
ratings = pd.read_csv("./ratings.csv",sep=",",names=["user","movie_id","rating"])
movies = pd.read_csv("./movies.csv",sep=",",names=["movie_id","movie_name"])
```

其中，sep 代表分隔符，name 代表每一列的字段名，返回的是类似二维表的 DataFrame 类型数据。

5.3.1 用户角度分析

首先可以使用 Pandas 的 head()函数查看 ratings 的结构，代码如下所示。head()是 DataFrame 的成员函数，用于返回前 n 行数据。其中，n 是参数，它代表选择的行数，默认值是 5。

```
>>> ratings.head()
```

输出为：

```
                    user                    movie_id    rating
0    0ab7e3efacd56983f16503572d2b9915       5113101       2
1    84dfd3f91dd85ea105bc74a4f0d7a067       5113101       1
2    c9a47fd59b55967ceac07cac6d5f270c       3718526       3
3    18cbf971bdf17336056674bb8fad7ea2       3718526       4
4    47e69de0d68e6a4db159bc29301caece       3718526       4
```

可以看到，用户 ID 是长度一致的字符串（实际是经过 MD5 处理的字符串），影片 ID 是数字。如果想查看一共有多少条数据，可以使用 rating.shape，输出为(1048575,3)。1048575 代表一共有 10 485 75 条数据，3 对应 3 列。

然后可以查看用户的评论情况，如数据中一共有多少人参与评论及每个人评论的次数。由于 ratings 数据中的每个用户都可以为多部电影进行评分，因此可以按用户进行分组，然后使用 count()统计数量。为了方便查看，可以对分组计数后的数据进行排序，再使用 head()函数查看排序后的情况，代码如下所示。其中 groupby 指按参数指定的字段进行分组，它可以有多个字段；count 是对分组后的数据进行计数；sort_values 则是按照某些字段的值进行排序；ascending=False 代表逆序。

```
>>> ratings_gb_user =
ratings.groupby('user').count().sort_values(by='movie_id',
ascending=False)
>>> ratings_gb_user.head()
```

输出为：

```
                    user                movie_id    rating
535e6f7ef1626bedd166e4dfa49bc0b4        1149        1149
```

```
425889580eb67241e5ebcd9f9ae8a465      1083       1083
3917c1b1b030c6d249e1a798b3154c43      1062       1062
b076f6c5d5aa95d016a9597ee96d4600      864        864
b05ae0036abc8f113d7e491f502a7fa8      844        844
```

可以看出，评分次数最多的用户 ID 是 535e6f7ef1626bedd166e4dfa49bc0b4，一共评论了 1149 次。这里 movie_id 和 rating 的数据是相同的，是因为其计数规则一致，属于冗余数据。因为 head()函数能看到的数据太少，所以可以使用 describe()函数查看统计信息，代码如下所示。

```
>>> ratings_gb_user.describe()
```

输出为：

```
              movie_id              rating
count       273826.000000       273826.000000
mean        3.829348            3.829348
std         14.087626           14.087626
min         1.000000            1.000000
25 %        1.000000            1.000000
50 %        1.000000            1.000000
75 %        3.000000            3.000000
max         1149.000000         1149.000000
```

从输出的信息中可以看出，一共有 273 826 个用户参与评分，用户评分的平均次数是 3.829 348 次。标准差是 14.087 626，相对来说还是比较大的。而从最大值、最小值和中位数可以看出，大部分用户对影片的评分次数还是很少的。

如果想更直观地查看数据的分布情况，可以查看直方图，代码如下所示。

```
>>> ratings_gb_user.movie_id.hist(bins = 50)
```

效果如图 5-2 所示。

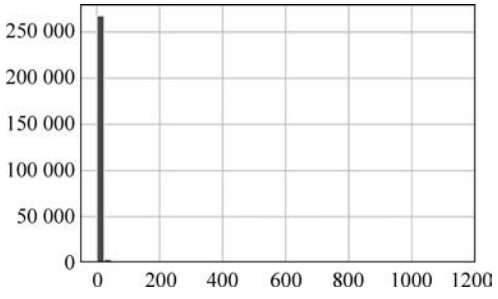

图 5-2　用户评分数据直方图

可以看出，大部分用户都集中在评分次数很少的区域，基本上没有大于 100 的数据。

如果想查看某一个区间的数据，可以使用 range 参数。例如，想看评论次数为 1～10 的用户分布情况，可以将参数 range 设置为[1, 10]，代码如下：

```
>>> ratings_gb_user.movie_id.hist(bins = 50)
```

结果如图 5-3 所示。

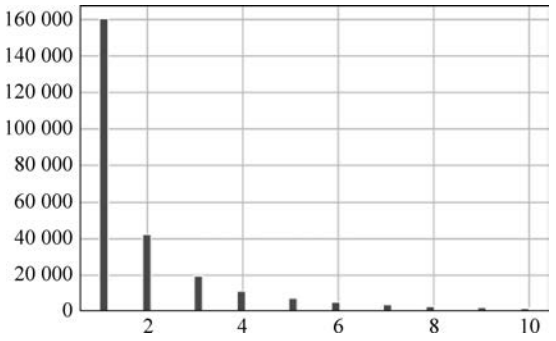

图 5-3　评论次数为 1～10 的用户分布情况

可以看到，无论是整体还是局部，评论次数越多，对应的用户数越少。结合之前的分析，大部分用户(75%)的评分次数都小于 4 次，这基本上符合常规的认知。

除了从评论次数上进行分析，也可以从评分值上进行统计，代码如下所示。其中，groupby 指按参数指定的字段进行分组，它可以有多个字段；count 是对分组后的数据进行计数；sort_values 是按照某些字段的值进行排序；ascending＝False 代表逆序。

```
>>> user_rating = ratings.groupby('user').mean().sort_values(by = 'rating', ascending =
False)
>>> user_rating.rating.describe()
```

输出为：

```
count        273826.00000
mean          3.439616
std           1.081518
min           1.000000
25 %          3.000000
50 %          3.500000
75 %          4.000000
max           5.000000
Name: rating, dtype: float64
```

可以看出，所有用户的评分的均值是 3.439616，而且大部分人(75%)的评分在 4 分

左右，所以整体的评分还是比较高的，说明用户对电影的态度并不是很苛刻，或者收集的数据中电影的总体质量不错。

接着可以将评分次数和评分值进行结合，从二维的角度进行观察，代码如下所示。其中，groupby 指按参数指定的字段进行分组，它可以有多个字段；count 是对分组后的数据进行计数；sort_values 是按照某些字段的值进行排序；ascending＝False 代表逆序。

```
>>> user_rating = ratings.groupby('user').mean().sort_values(by = 'rating', ascending =
False)
>>> ratings_gb_user = ratings_gb_user.rename(columns = {'movie_id_x':'movie_id','rating_y':
'rating'})
>>> ratings_gb_user.plot(x = 'movie_id', y = 'rating', kind = 'scatter')
```

结果如图 5-4 所示。

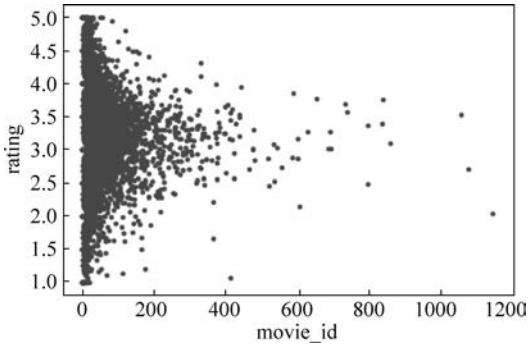

图 5-4　结合评分次数和评分值

可以看到，分布基本上呈“＞”形状，大部分用户评分较少，且中间分数的用户偏多。

5.3.2　电影角度分析

接下来，可以用相似的方法从电影的角度查看数据的分布情况，如每一部电影被评分的次数。要获取每一部电影的被评分次数就需要对影片的 ID 进行分组和计数。为了提高数据的可观性，可以通过关联操作显示影片的名称，使用 Pandas 的 merge 函数可以很容易实现，代码如下所示。在 merge 函数中，参数 how 代表关联的方式，如 inner 是内关联，left 是左关联，right 代表右关联；on 是关联时使用的键名，由于 ratings 和 movies 对应的电影的字段名是一样的，因此可以只传入 movie_id 这一个参数，否则需要使用 left_on 和 right_on 参数。

```
>>> ratings_gb_movie = ratings.groupby('movie_id').count().sort_values(by = 'user',
ascending = False)
>>> ratings_gb_movie = pd.merge(ratings_gb_movie,movies, how = 'left', on = 'movie_id')
>>> ratings_gb_movie.head()
```

输出为：

```
      movie_id     user rating    movie_name
0     3077412      320  320       寻龙诀
1     1292052      318  318       肖申克的救赎 - 电影
2     25723907     317  317       捉妖记
3     1291561      317  317       千与千寻
4     2133323      316  316       白日梦想家 - 电影
```

可以看到，被评分次数最多的电影是《寻龙诀》，一共被评分 320 次。同样，user 和 rating 的数据是一致的，属于冗余数据。下面查看详细的统计数据，代码如下所示。

```
>>> ratings_gb_movie.user.describe()
```

输出为：

```
count    22847.00000
mean        45.895522
std         61.683860
min          1.000000
25%          4.000000
50%         17.000000
75%         71.000000
max        320.000000
```

可以看到，一共有 22 847 部电影被用户评分，平均被评分次数接近 46，大部分影片（75%）的被评分次数在 71 次左右。

接着查看直方图，代码如下所示。

```
>>> ratings_gb_user.movie_id.hist(bins = 50)
```

结果如图 5-5 所示。

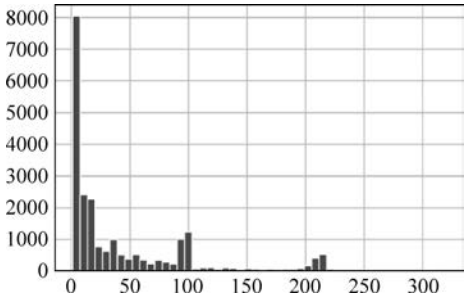

图 5-5　电影被评分次数直方图

可以看到，大约被评分 80 次之前的电影数，基本上是随着评论次数的增加在减少，但是被评论 100 次和 200 次左右的影片却有异常的增加。此外，可以看到分布的标准差比较大，从而得知数据质量并不是太高，但整体上的趋势还是基本符合常识。

接下来，同样要对评分值进行观察，代码如下所示。

```
>>> movie_rating = ratings.groupby('movie_id').mean().sort_values(by = 'rating', ascending = False)
>>> movie_rating.describe()
```

输出为：

```
count    22847.000000
mean         3.225343
std          0.786019
min          1.000000
25 %         2.800000
50 %         3.333333
75 %         3.764022
max          5.000000
```

从统计数据中可以看出，所有电影的平均分数和中位数很接近，大约是 3.3，说明整体的分布比较均匀。

然后将结合被评分次数和评分值进行分析，代码如下所示。

```
>>> ratings_gb_movie = pd.merge(ratings_gb_movie, movie_rating, how = 'left', on = 'movie_id')
>>> ratings_gb_movie.head()
```

输出为：

```
      movie_id      user      rating_x  movie_name            rating_y
0     3077412       320         320     寻龙诀                3.506250
1     1292052       318         318     肖申克的救赎 - 电影   4.672956
2     25723907      317         317     捉妖记                3.192429
3     1291561       317         317     千与千寻              4.542587
4     2133323       316         316     白日梦想家            3.990506
```

从输出的数据可以看出，有些电影（如《寻龙诀》）虽然本身被评分的次数很多，但是综合评分并不高，这也符合实际的情况。

查看散点图，代码如下所示。

```
>>> ratings_gb_movie.plot(x = 'user', y = 'rating', kind = 'scatter')
```

结果如图 5-6 所示。

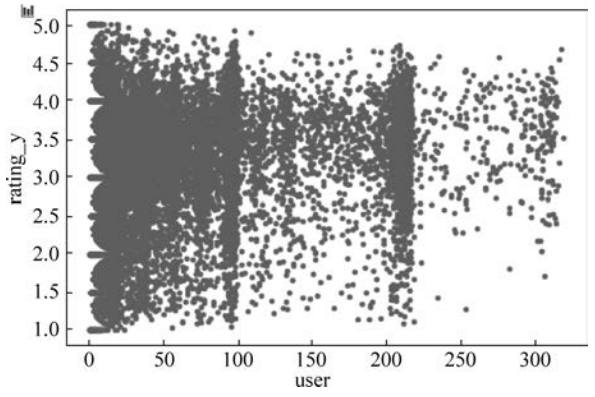

图 5-6　结合被评分次数和评分值的散点图

可以看到，总体上数据还是呈现“>”分布，但是在被评分次数为 100 次和 200 次左右出现了比较分散的情况，这和图 5-5 是相对应的。这也许是一种特殊现象，而是否是一种规律就需要更多的数据来分析和研究。

当前的分析结果可以有较多用途，如做一个观众评分量排行榜或者电影评分排行榜等。结合电影标签就可以做用户的兴趣分析。

5.4　电影推荐

在对数据有足够的认知之后，可以根据当前数据给用户推荐其没有看过但是很有可能会喜欢的电影。推荐算法大致可以分为三类：协同过滤推荐算法、基于内容的推荐算法和基于知识的推荐算法。其中，协同过滤推荐算法是诞生较早且较为著名的算法，其通过对用户历史行为数据的挖掘发现用户的偏好，基于不同的偏好对用户进行群组划分并推荐品味相似的商品。

协同过滤推荐算法分为两类，分别是基于用户的协同过滤算法（user-based collaborative filtering）和基于物品的协同过滤算法（item-based collaborative filtering）。基于用户的协同过滤算法是通过用户的历史行为数据发现用户对商品或内容的喜好（如商品购买、收藏、内容评论或分享），并对这些喜好进行度量和打分。根据不同用户对相同商品或内容的态度和偏好程度计算用户之间的关系，然后在有相同喜好的用户间进行商品推荐。其中，比较重要的就是距离的计算，可以使用余弦相似性、Jaccard 实现。整体的实现思路是：使用余弦相似性构建邻近性矩阵，再使用 KNN 算法从邻近性矩阵中找到某用户邻近的用户，并将这些邻近用户点评过的电影作为备选，然后将邻近性的值作为推荐的得分，相同的分数可以累加，最后排除该用户已经评价过的电影。部分脚本如下所示。

```
# 根据余弦相似性建立邻近性矩阵
ratings_pivot = ratings.pivot('user','movie_id','rating')
ratings_pivot.fillna(value = 0)
```

```
m,n = ratings_pivot.shape
userdist = np.zeros([m,m])
for i in range(m):
    for j in range(m):
        userdist[i,j] = np.dot(ratings_pivot.iloc[i,],ratings_pivot.iloc[j,]) \
        /np.sqrt(np.dot(ratings_pivot.iloc[i,],ratings_pivot.iloc[i,])\
         * np.dot(ratings_pivot.iloc[j,],ratings_pivot.iloc[j,]))
proximity_matrix = pd.DataFrame(userdist,index = list(ratings_pivot.index),columns = list
(ratings_pivot.index))

# 找到邻近的 k 个值
def find_user_knn(user, proximity_matrix = proximity_matrix, k = 10):
    nhbrs = userdistdf.sort(user,ascending = False)[user]1:k + 1]
    #在一列中降序排序,除去第一个(自己)后为近邻
    return nhbrs

# 获取推荐电影的列表
def recommend_movie(user, ratings_pivot = ratings_pivot, proximity_matrix = proximity_
matrix):
    nhbrs = find_user_knn(user, proximity_matrix = proximity_matrix, k = 10)
    recommendlist = {}
    for nhbrid in nhbrs.index:
        ratings_nhbr = ratings[ratings['user'] == nhbrid]
        for movie_id in ratings_nhbr['movie_id']:
            if movie_id not in recommendlist:
                recommendlist[movie_id] = nhbrs[nhbrid]
            else:
                recommendlist[movie_id] = recommendlist[movie_id] + nhbrs[nhbrid]
    # 去除用户已经评分过的电影
    ratings_user  = ratings[ratings['user'] == user]
    for movie_id in ratings_user['movie_id']:
        if movie_id in recommendlist:
            recommendlist.pop(movie_id)
    output = pd.Series(recommendlist)
    recommendlistdf = pd.DataFrame(output, columns = ['score'])
    recommendlistdf.index.names = ['movie_id']
    return recommendlistdf.sort('score',ascending = False)
```

建立邻近性矩阵是很消耗内存的操作，如果执行过程中出现内存错误，则需要换用内存更大的机器运行，或者对数据进行采样处理，从而减少计算量。代码中给出的是基于用户的协同过滤算法，读者可以尝试写出基于电影的协同过滤算法，然后对比算法的优良性。