第 13 章 大数据分析最佳实战—电影推荐系统

★本章导读★

本章将通过完整的实战项目——电影推荐系统，将本书所涉及的 DataFrame 处理与分析方法以及 Pyspark ML 机器学习方法进行深度结合，实现大数据分析处理的真谛。同时，本章会对推荐系统进行详细介绍，使读者深入了解业务场景，在实际生产中同样如此，首先需要对业务场景有更为全面的认识，才能够选择合理的方法实现最佳效果。

★知识要点★

通过本章内容的学习，读者将掌握以下知识：

- 推荐系统常见推荐方法
- ALS 算法原理
- 熟练完成电影推荐系统项目

13.1 推荐系统介绍

随着互联网的飞速发展，更多的实体店转向了在线销售的网店，不同于线下实体店的是，在实体店购物时，通常会有导购员或者销售为顾客推荐适合的商品，而网上销售则没有导购员，这便给用户带来了困扰，用户不知如何从琳琅满目的商品中选择自己需要的，或者会花费大量时间在选购上，我们必须自行检索，为用户选择最合适的商品，这种检索、展示的功能变得尤为重要。

13.1.1 推荐系统

推荐系统的主要功能是用于将合适的商品或者内容推荐给用户，这有很多种方法，如热门商品展示、新品推荐等等，推荐系统则是以个性化的方式推荐合适的商品给和合适的用户，从而增强用户体验和用户粘粘性。面对大量商品和海量数据，用户指定个性化的方案然后根据用户喜好推荐商品。

推荐有利于帮助用户快速检索并展示出用户可能喜欢的信息或者商品。推荐系统可以用于多种场景，例如网店、商品零售、多媒体资源（电影/音乐/电视剧/书籍）、广告等等。

从用户角度来说，推荐系统可以帮助用户节省大量的时间，同时可以挖掘用户潜在喜好，增强用户体验，为用户提供足够吸引力的内容。

从企业角度来说，推荐系统可以帮助企业带来更高的收益，增强用户的购买欲望，同时通过挖掘用户喜好可以发现当前市场趋势，及时做出战略性调整等。

13.1.2 推荐系统需求

本章节实现的推荐系统为电影推荐系统，即为用户推荐可能喜欢的电影。本章是通过用户对电影的观看评分数据，挖掘相似用户和相似电影信息，为用户提供未观看过的优质电影。俗话说“物以类聚人以群分”，本章节项目实战就是通过大数据分析处理的方式对这句话的完美诠释。

13.2 常见推荐系统的方法

13.2.1 基于内容推荐系统

基于内容的推荐系统即为，通过挖掘用户过去的喜好向他们推荐商品或信息。基于内容的推荐系统的核心思想就是挖掘商品/信息之间的相速度，并且基于用户的偏好数据向用户推荐商品/信息。

批注 [雷1]: 语意不明，需修改

基于内容的推荐系统的重要环节就是要大量收集用户和商品/信息的资料数据，并整理数据相关的属性信息，例如，一部电影可能包含的属性信息如下：

电影 id	动作	喜剧	爱情	战争	剧情
1235	0.2	0.6	0.3	0.2	0.4

上面是一部电影的特征信息，通过该方式大量收集商品信息并处理生成如上条目所展示的电影信息内容，就形成了商品信息表。

除了商品信息数据还远远不够，我们还需要大量收集用户行为日志，通过对日志分析，形成用户信息表，从而了解用户的喜好信息，如通过分析某用户看过的 1000 部电影，可以了解用户对不同类型电影的观看数据，从而可知用户更喜好的电影类型，如下所示：

用户 id	动作	喜剧	爱情	战争	剧情
U1536	0.1	0.7	0.1	0.0	0.2

从上数据可以直观的看出，该用户可能对喜剧类型的电影偏好程度较高。我们可以通过计算相似度的方法为用户进行推荐，即用户与电影直接的相似度分数越高，则用户可能喜欢该电影的可能性就越大。通常，计算相似度的方法有欧式距离和余弦相似度，下面分别对两种计算方法进行介绍。

1. 欧式距离

欧式距离即计算两个向量之间的距离，如上述电影和用户的特征数据可以看做两个 n 维向量，这里 n 为 5，使用欧式距离公式可以方便的计算两个 n 维向量之间的距离。欧氏距离公式为：

$$d(x,y) = \sqrt{(x_1-y_1)^2+(x_2-y_2)^2+\cdots+(x_n-y_n)^2}$$

两个向量的距离越大则代表两个向量之间的相似度越低。

2. 余弦相似度

余弦相似度也是常用的计算两个向量之间相似度的方法，与欧式距离不同的是，余弦相似度计算的是两个向量直之间夹角的余弦值，值为 0.0~1.0，值越小则代表两向量之间的相似度越高，余弦相似度的计算公式为：

$$\cos(x,y) = \frac{x*y}{|x|*|y|}$$

基于内容的推荐系统的优点在于其原理非常简单，不需要复杂的数据处理和计算，仅仅基于用户和物品的评分信息就可以进行推荐。但是，基于内容的推荐系统的缺点也是存在的，基于内容可能没有办法很准确的反映商品的属性，所以导致推荐的商品不够个性化。

13.2.2 协同过滤推荐系统

协同过滤是推荐系统中一大类算法，包括如 userCF、itemCF 等等。协同过滤简单来说是利用某兴趣相投、拥有共同经验之群体的喜好来推荐用户感兴趣的信息，个人通过合作的机制给予信息相当程度的回应（如评分）并记录下来以达到过滤的目的进而帮助别人筛选信息，回应不一定局限于特别感兴趣的，特别不感兴趣信息的纪录也相当重要。

与传统的基于内容过滤直接分析内容进行推荐不同，协同过滤分析用户兴趣，在用户群中找到指定用户的相似（兴趣）用户，综合这些相似用户对某一信息的评价，形成系统对该指定用户对此信息的喜好程度预测。

与传统文本过滤相比，协同过滤有下列优点：

1. 能够过滤难以进行机器自动基于内容分析的信息。如艺术品、音乐;
2. 能够基于一些复杂的，难以表达的概念（信息质量、品位)进行过滤;
3. 推荐的新颖性。

正因为如此，协同过滤在商业应用上也取得了不错的成绩。Amazon，CDNow，MovieFinder 都采用了协同过滤的技术来提高服务质量。

缺点是：

1. 用户对商品的评价非常稀疏，这样基于用户的评价所得到的用户间的相似性可能不准确（即稀疏性问题）;
2. 随着用户和商品的增多，系统的性能会越来越低;
3. 如果从来没有用户对某一商品加以评价，则这个商品就不可能被推荐（即最初评价问题）。

协同过滤其核心在于用户和物品之间关联的用户行为数据，协同过滤可以分为基于近邻的协同过滤和基于模型的协同过滤。

1. 基于近邻的协同过滤：

（1） 基于用户（UserCF）

（2） 基于物品（ItemCF）

基于用户的协同过滤（UserCF）的基本原理是根据所有用户对物品的偏好，发现与当前用户口味和偏好相似的“邻居”用户群，并推荐近邻所偏好的物品。在一般的应用中是采用计算“K- 近邻”的算法；基于这 K 个邻居的历史偏好信息，为当前用户进行推荐。基于用户的协同过滤机制是在用户的历史偏好的数据上计算用户的相似度，它的基本假设是，喜欢类似物品的用户可能有相同或者相似的口味和偏好，如图 13-1 所示。

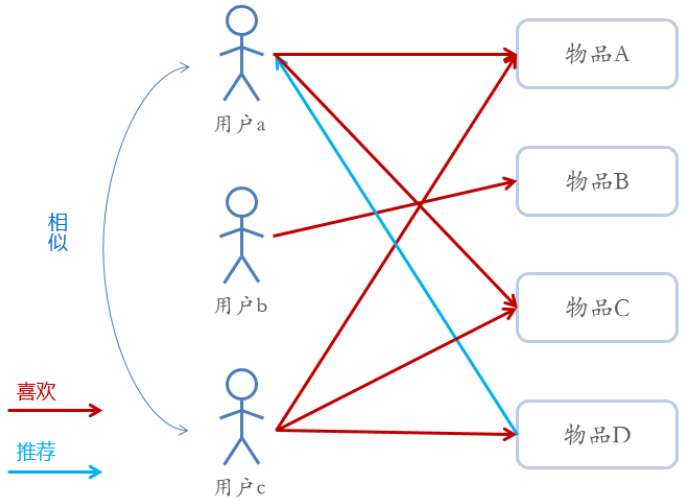

图 13-1 基于用户的协同过滤

基于物品的协同过滤推荐的基本原理与基于用户的类似，只是使用所有用户对物品的偏好，发现物品和物品之间的相似度，然后根据用户的历史偏好信息，将类似的物品推荐给用户，如图 13-2 所示。

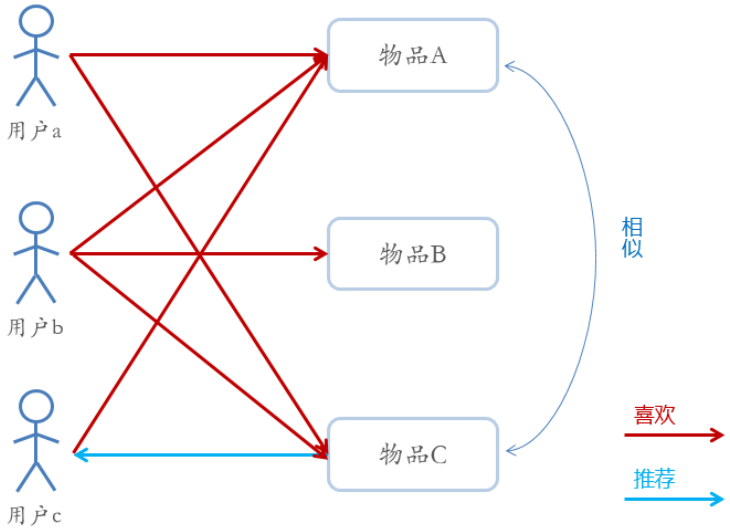

图 13-2 基于用户的协同过滤

同样是协同过滤，在 UserCF 和 ItemCF 两个策略中应该如何选择呢？

基于物品的协同过滤（ItemCF）适用于物品的个数是远远小于用户的数量，而且物品的个数和相似度相对比较稳定，同时基于物品的机制比基于用户的实时性更好一些。而 UserCF 应用在物品数量可能远大于用户的个数，而且物品的更新程度也很快，所以它的相似度依然不稳定，这时用 UseCF 可能效果更好。所以，推荐策略的选择其实和具体的应用场景有很大的关系。

2. 基于模型的协同过滤：

（1） 奇异矩阵分解（SVD）

（2） 潜在语义分析（LSA）

（3） 支持向量机（SVM）

基于模型的协同过滤的基本思想：用户具有一定的特征，决定着他的偏好选择；物品具有一定的特征，影响着用户是否选择它；用户之所以选择某一个商品，是因为用户特征与物品特征相互匹配。基于模型的协同过滤推荐，就是基于样本的用户偏好信息，训练一个推荐模型，然后根据实时的用户喜好的信息进行预测新物品的得分，然后通过计算进行推荐。基于模型的协同过滤方法是要用户/物品偏好数据来训练模型，挖掘内在规律，再用模型来做预测。训练模型时，可以基于标签内容来提取物品特征，也可以让模型去发掘物品的潜在特征，这样的模型被称为隐语义模型（Latent Factor Model，LFM），上述提到的三种模型均属于隐语义模型的实例代表。由于基于模型的方法数学推导过程较为复杂，本小节不再过多的展开叙述，感兴趣的读者可以自行研究。

本章节的实战使用的是奇异矩阵分解（SVD）的方法，具体来说，模型求解使用的算法为 ALS，所以这里对奇异矩阵分解（SVD）的原理进行简单介绍。

假设用户物品评分矩阵为 R，现在有 m 个用户，n 个物品，我们想要发现 k 个隐类（用户和物品的隐藏特征），我们的任务就是找到两个矩阵 P 和 Q，使这两个矩阵的乘积近似等于 R，即将用户物品评分矩阵 R 分解成为两个低维矩阵相乘，如图 13-3 所示。

$$P_{m\times k}^{T}\cdot Q_{k\times n}=R_{m\times n}\approx R_{m\times n}$$

图 13-3 奇异矩阵分解

13.2.3 混合推荐系统

机器学习中有所谓的集成学习（Ensemble Learning），本质上是利用多个分类或者回归算法，通过这些算法的有效整合获得更好的分类或者预测效果。集成方法之所以有效，是因为通过不同的算法组合可以有效地降低系统性误差（方差），最终达到更好的效果。

混合推荐系统的思路跟上面的介绍如出一辙。古话说”三个臭皮匠顶个诸葛亮“，我想用这句话来形容混合推荐算法是非常恰当的。混合推荐算法就是利用两种或者两种以上推荐算法来配合，克服单个算法存在的问题，期望更好地提升推荐的效果，如图 13-4 所示。

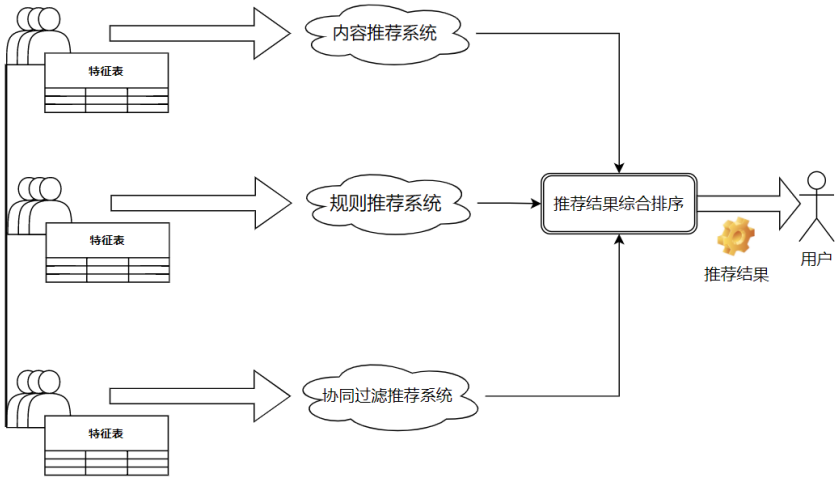

图 13-4 混合推荐系统

如上图所示，混合推荐系统可以包含基于内容的推荐系统、基于协同过滤的推荐系，以及基于规则的推荐系统等等。随着数据呈现多元化的趋势，混合推荐系统已经变成各种业务场景中不可或缺的部分，它可以帮助企业精准的把控用户的消费习惯，为用户提供合适的内容，为企业带来更大的利益和回报。

13.3 电影推荐系统实战

13.3.1 数据集介绍

本节将展开电影推荐系统实战的完整序幕。首先对本节使用的数据集进行简单介绍。本实战使用的是推荐算法常用的数据集 movie_ratings_df.csv。数据集文件为 csv 格式，数据内容如图 13-5 所示。

userId	title	rating
82	Zeus and Roxanne (1997)	2
181	Zeus and Roxanne (1997)	3
463	Zeus and Roxanne (1997)	1
792	Zeus and Roxanne (1997)	3
798	Zeus and Roxanne (1997)	3
881	Zeus and Roxanne (1997)	1
7	Young Poisoner's Handbook, The (1995)	3
13	Young Poisoner's Handbook, The (1995)	1
21	Young Poisoner's Handbook, The (1995)	2
49	Young Poisoner's Handbook, The (1995)	5

图 13-5 查看数据集内容

数据文件包含 3 列，分别是 userId、title、rating，userId 指用户 id，这里使用的是数字 id，title 是指电影名称，rating 是指用户对该电影的评分。例如：userId 为 82 的用户为电影《Zeus and Roxanne (1997)》

打分为 2。评分可以直接反应用户对不同电影的喜好程度。

13.3.2 读取数据集

步骤 1：启动 Hadoop 集群

打开虚拟机 master、slave1、slave2 三台机器，并在 master 节点终端下输入如下命令启动 hadoop，按前面章节的方法进行验证集群是否正常启动。

```
cd  /usr/local/src/hadoop-bin-2.6.5/sbin
./start-all.sh
```

步骤 2：新建 Notebook 文件

通过右上角【New】下拉菜单选择【Python3】，并确定，完成 Notebook 的新建工作，如图 13-6 所示。

图 13-6 新建 Notebook 文件

步骤 3:重命名 Notebook

单击【Untitled】，输入“Chapter08”，并确认，为 Notebook 重命名，如图 13-7 所示。

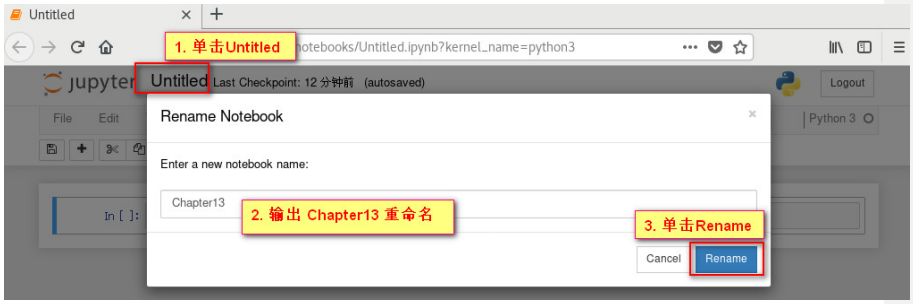

图 13-7 重命名 Notebook 文件

步骤 3：上传文件到 HDFS

在 master 节点终端下，进入到数据集所在目录，本书将数据集文件放于 Jupyter Notebook 的数据目录 data 下，使用 HDFS 命令上传数据集文件，代码如下所示。

```
cd  /root/pyspark/
hdfs  dfs -put  movie_ratings_df.csv  /data/
```

接下来，通过 HDFS Web 页面查看文件是否上传成功，如下图 13-8 所示。

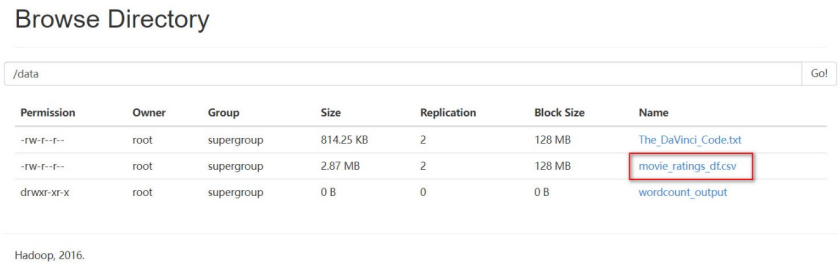

图 13-8 查看 HDFS 数据集文件

步骤 4：创建 SparkSession

在完成上述步骤之后，在 Notebook 文件中输入如下代码，创建本章节实例的 SparkSession，如下图 13-9 所示。

```
In [6]: from pyspark.sql import SparkSession
spark = SparkSession\
    .builder\
    .appName('Recommend-System-Pyspark')\
    .getOrCreate()

In [7]: spark

Out[7]: SparkSession - hive
SparkContext

Spark UI
Version
v2.4.3
Master
local[*]
AppName
Recommend-System-Pyspark
```

图 13-9 创建 SparkSession

步骤 5：读取数据集

接下来使用 spark.read.csv 方法，读取数据集，注意这时我们读取的文件为 HDFS 路径。读取代码如下图 13-10 所示。

```
In [10]: df = spark.read.csv('/data/movie_ratings_df.csv',
                             inferSchema=True,
                             header=True)
```

图 13-10 读取数据集

我们可以通过 count()函数查看数据集的大小，如图 13-11 所示。

```
In [12]: print((df.count(),len(df.columns)))
(100000, 3)
```

图 13-11 查看数据集量大小

接下来，我们打印数据集的字段结构，如图 13-12 所示。

```
In [14]: df.printSchema()
```

```
root
 |— userId: integer (nullable = true)
 |— title: string (nullable = true)
 |— rating: integer (nullable = true)
```

图 13-12 打印数据结构

13.3.3 探究数据集

在本小节，我们将从不同维度查看数据集的内容，对数据集进行简单的探索。

步骤 1：打印数据集

我们使用 rand()函数随机顺序打印原始数据集，查看读取数据的正确性，代码如图 13-13 所示。

```
In [16]: df.orderBy(rand()).show(10,False)
```

随机排序并打印

```
+------+-----------------------------------------+------+
|userId|title                                    |rating|
+------+-----------------------------------------+------+
|544   |Jackie Brown (1997)                      |4     |
|884   |Fargo (1996)                             |5     |
|38    |Jury Duty (1995)                         |1     |
|367   |Jaws (1975)                              |4     |
|286   |Tommy Boy (1995)                         |1     |
|882   |Terminator, The (1984)                   |5     |
|200   |Pulp Fiction (1994)                      |4     |
|813   |Beautician and the Beast, The (1997)|3     |
|226   |Unforgiven (1992)                        |5     |
|638   |Full Metal Jacket (1987)                 |3     |
+------+-----------------------------------------+------+
only showing top 10 rows
```

图 13-13 打印数据集

步骤 2：查看用户观看电影数

我们使用 groupBy 算子按用户 Id 进行分组统计，查看用户观看电影数情况，这里展示前 10 行数据，如图 13-14 所示。

```
In [18]: df.groupBy('userId')\
             .count()\
             .orderBy('count',ascending=False)\
             .show(10,False)
```

```
+------+-----+
|userId|count|
+------+-----+
|405   |737  |
|655   |685  |
|13    |636  |
|450   |540  |
|276   |518  |
|416   |493  |
|537   |490  |
|303   |484  |
|234   |480  |
|393   |448  |
+------+-----+
only showing top 10 rows
```

图 13-14 用户观看电影数统计

步骤 3：查看电影被观看数

同样，使用 groupBy 算子按 title 进行分组统计，查看每部电影被观看数据量情况的统计，这里同样展示前 10 行数据，如图 13-15 所示。

```
In [19]: df.groupBy('title')\
             .count()\
             .orderBy('count',ascending=False)\
             .show(10,False)
```

```
+---------------------------------+-----+
|title                            |count|
+---------------------------------+-----+
|Star Wars (1977)                 |583  |
|Contact (1997)                   |509  |
|Fargo (1996)                     |508  |
|Return of the Jedi (1983)        |507  |
|Liar Liar (1997)                 |485  |
|English Patient, The (1996)      |481  |
|Scream (1996)                    |478  |
|Toy Story (1995)                 |452  |
|Air Force One (1997)             |431  |
|Independence Day (ID4) (1996)    |429  |
+---------------------------------+-----+
only showing top 10 rows
```

图 13-15 查看电影被观看数

13.3.4 特征工程构建

接下来使用 StringINdexer 和 ML Pipeline 对数据集进行特征工程的构建，目的是为了构建可以用于后续模型训练的数据集。StringIndexer 会将字符串特征编码成数字特征。对于电影推荐系统数据集，这里只有 title 一个字符串特征，所以转换的列只有这一列即可。

步骤 1：导入相关库

这里导入的特征处理的库包括 StrinIndex 和 IndexToString，以及 Spark ML 中的 Pipeline。

```
In [30]: from pyspark.ml.feature import StringIndexer,IndexToString
         from pyspark.ml import Pipeline
```

批注 [雷2]: 对图片内容进行讲解

图 13-16 导入特征构建库

步骤 2：创建 StringIndexer Pipeline

创建 StringIndexer 的方法请参考第 12 章节中所讲解内容。

```
In [52]: stringIndexer = StringIndexer(
             inputCol="title",
             outputCol="title_new")

In [66]: pipeline = Pipeline(stages = [stringIndexer])
```

批注 [雷3]: 对图片内容进行讲解

批注 [HZ4R3]: 图 13-16 和图 13-17 中使用的代码在第 12 章节已经详细介绍过作用和使用方法，这一章是对前面章节涉及的内容的综合应用，这里再讲解一遍是不是和前面章节重复一样了就？

图 13-17 创建 StringIndexer Pipeline

完成创建之后可以通过 getStages()函数查看 Pipeline 的内容，如图 13-18 所示。

```
In [138]: pipeline.getStages()
Out[138]: [StringIndexer_da1ddbecd0d3]
```

图 13-18 创建 Pipeline Stages

步骤 3：特征转换

使用 fit()和 transform()函数对原始数据集进行特征转换，代码如下图 13-19 所示。

```
In [157]: pipelineModel = pipeline.fit(df)

In [158]: index_df = pipelineModel.transform(df)
```

图 13-19 特征转换代码

步骤 4：查看数据集

在完成上述步骤的转换之后，我们需要查看转换后数据集的结构以及内容从而判断是否正确转换，如图 13-20、13-21 所示。

```
In [72]: index_df.printSchema()
root
 |-- userId: integer (nullable = true)
 |-- title: string (nullable = true)
 |-- rating: integer (nullable = true)
 |-- title_new: double (nullable = false)
```

图 13-20 查看数据集结构

我们通过 show()方法将转换后的数据集的前 10 行进行打印，如图 13-21 所示。

```
In [71]: index_df.show(10, False)
+------+-------------+------+---------+
|userId|title        |rating|title_new|
+------+-------------+------+---------+
|196   |Kolya (1996)|3     |287.0    |
|63    |Kolya (1996)|3     |287.0    |
|226   |Kolya (1996)|5     |287.0    |
|154   |Kolya (1996)|3     |287.0    |
|306   |Kolya (1996)|5     |287.0    |
|296   |Kolya (1996)|4     |287.0    |
|34    |Kolya (1996)|5     |287.0    |
|271   |Kolya (1996)|4     |287.0    |
|201   |Kolya (1996)|4     |287.0    |
|209   |Kolya (1996)|4     |287.0    |
+------+-------------+------+---------+
only showing top 10 rows
```

图 13-21 查看数据集前 10 行

接下来，我们同样按照编码后的 title_new 字段进行统计，并从高到底排列，附带数字编码与原始字符串特征的映射关系，从而判断数据内容是否完全一致，如图 13-22 所示。

```
In [161]: index_df\
              .groupBy('title_new')\
              .count()\
              .orderBy('count',ascending=False)\
              .show(10,False)
```

```
+---------+-----+
|title_new|count|
+---------+-----+
|0.0      |583  |
|1.0      |509  |
|2.0      |508  |
|3.0      |507  |
|4.0      |485  |
|5.0      |481  |
|6.0      |478  |
|7.0      |452  |
|8.0      |431  |
|9.0      |429  |
+---------+-----+
only showing top 10 rows
```

title_new 为 0.0 的电影数据总数为583条

```
In [165]: index_df.select("title", "title_new")\
              .distinct()\
              .orderBy("title_new",ascending=True)\
              .show(10,False)
```

```
+---------------------------------+---------+
|title                            |title_new|
+---------------------------------+---------+
|Star Wars (1977)                 |0.0      |
|Contact (1997)                   |1.0      |
|Fargo (1996)                     |2.0      |
|Return of the Jedi (1983)        |3.0      |
|Liar Liar (1997)                 |4.0      |
|English Patient, The (1996)      |5.0      |
|Scream (1996)                    |6.0      |
|Toy Story (1995)                 |7.0      |
|Air Force One (1997)             |8.0      |
|Independence Day (ID4) (1996)    |9.0      |
+---------------------------------+---------+
only showing top 10 rows
```

《Star Wars（1977）》编码为0.0

图 13-22 统计并验证数据集

通过观察可知，《Star Wars（1977）》被编码为 0.0，结合上图和图 13-14，title 为《Star Wars (1977)》的电影和 title_name 为 0.0 对应的的电影数据量均为 583，可以验证特征工程构建的正确性。

13.3.5 创建训练数据集

在上一小节中我们完成了对原始数据集的特征转换，构建了特征工程，本小节将对数据集进行切分，用于后续模型的训练和测试环节，我们对数据集按 0.75:0.25 的比例切分为训练集和测试集，当然读者也可以自行选择其他比例进行切分，如 0.7:03，具体实现代码如图 13-23 所示。

```
In [81]: train_df, test_df = index_df.randomSplit([0.75, 0.25])
```

```
In [82]: print("Train Dataset Num :", train_df.count())
         print("Test Dataset Num :", test_df.count())
```

```
Train Dataset Num : 75125
Test Dataset Num : 24875
```

图 13-23 切分数据集

13.3.6 ALS 模型原理介绍

本小节我们将创建 ALS 模型，对训练集进行训练，用于构建电影推荐系统模型，我们需要从 Pyspark ML 库中导入 ALS 模型。

下面对 ALS 算法进行简单介绍。ALS 属于协同过滤算法中的一种，通常用于推荐系统，例如电影推荐，商品推荐系统，广告推荐系统等。

ALS 是交替最小二乘（alternating least squares）的简称。在机器学习中，ALS 特指使用交替最小二乘求解的一个协同推荐算法。它通过观察到的所有用户给商品的打分，来推断每个用户的喜好并向用户推荐适合的商品。举个例子，我们看如图 13-24，一个 8x8 的用户打分矩阵。

			5				1
	2				3		
6		5					
				6			
		7				6	
				?		9	
			4		2		
9						3	

图 13-24 ALS 评分矩阵

这个矩阵的每一行代表一个用户(u1,u2,…,u8）、每一列代表一个商品(v1,v2,…,v8)、用户的打分为 1-9 分。那么问题来了：用户 u6 给物品 v5 的打分大概会是多少？如果我们不添加任何条件的话，打分之间是相互独立的，我们没有任何依据来推断 u6 给 v5 的打分，所以在这个打分矩阵的基础上，我们需要提出一个限制其自由度的合理假设，使得我们可以通过观察已有打分猜测未知打分。

ALS 的核心就是这样一个假设：打分矩阵是近似低秩的。换句话说，就是一个$U \times I$的打分矩阵可以由分解的两个小矩阵$U(m \times k)$和$I(k \times n)$的乘积来近似。这就是 ALS 的矩阵分解方法。ALS 算法相关数学推导较为繁琐，限于篇幅这里不再展开赘述。

13.3.7 模型训练与评估

本小节我们将创建 ALS 模型，对训练集进行训练，用于构建电影推荐系统模型，我们需要从

Pyspark ML 库中导入 ALS 模型。

步骤 1：创建 ALS 模型

ALS 模型位于 pyspark.ml.recommendation 下，创建 ALS 模型的方法同第 12 章节中其他机器学习模型的方法相同，这里我们对 ALS 模型的主要参数进行介绍。

- maxIter ： 算法运行的最大迭代次数（默认为 10）；
- regParam ： 指定 ALS 中的睁着参数（默认为 1.0）；
- userCol ： 指定 user 的列名；
- itemCol ： 指定 item 的列名；
- ratingCol ： 指定评分列的列名；
- nonegative ： 是否最最小二乘法使用非负约束（默认为 False）；
- coldStartStategy ： 设置冷启动策略。

需要注意的是，对于该模型，我们设置了两个比较重要的参数，分别是 nonnegative=True 和 coldStartStrategy=”drop”，其中前者是使模型不会在推荐系统中创建负数评分，后者是为了防止模型生成任何 NaN 评分的预测。

```
In [79]: from pyspark.ml.recommendation import ALS

In [78]: als = ALS(maxIter=10,
                   regParam=0.01,
                   userCol='userId',
                   itemCol='title_new',
                   ratingCol='rating',
                   nonnegative=True,
                   coldStartStrategy="drop")
```

批注 [雷5]: 需对图片进行讲解

图 13-25 创建 ALS 模型

步骤 2：训练 ALS 模型并预测

接下来我们使用上一步中创建的 ALS 模型进行训练，并在测试集上进行预测，如图 13-26 所示。

```
In [83]: rec_model = als.fit(train_df)

In [84]: predicted_df = rec_model.transform(test_df)
```

图 13-26 创建模型并预测

步骤 3：查看结果

在完成模型训练之后，同样，我们可以通过打印数据结构和内容两个维度进行查看，如图 13-27 所示。

```
In [85]: predicted_df.printSchema()

root
 |-- userId: integer (nullable = true)
 |-- title: string (nullable = true)
 |-- rating: integer (nullable = true)
 |-- title_new: double (nullable = false)
 |-- prediction: float (nullable = false)
```

```
In [87]: predicted_df.orderBy(rand()).show(10, False)

+------+----------------------------------------------+------+---------+----------+
|userId|title                                         |rating|title_new|prediction|
+------+----------------------------------------------+------+---------+----------+
|213   |Sting, The (1973)                             |4     |75.0     |4.394649  |
|499   |Alien (1979)                                  |4     |44.0     |3.3748245 |
|189   |Ben-Hur (1959)                                |4     |270.0    |4.327587  |
|294   |Romy and Michele's High School Reunion (1997)|2     |339.0    |3.3786664 |
|410   |Postman, The (1997)                           |3     |552.0    |4.132841  |
|416   |Manchurian Candidate, The (1962)              |5     |239.0    |4.8652353 |
|910   |Twister (1996)                                |3     |42.0     |2.8483937 |
|347   |Young Frankenstein (1974)                     |2     |117.0    |4.493652  |
|855   |Wings of Desire (1987)                        |4     |562.0    |3.8445659 |
|925   |Mary Shelley's Frankenstein (1994)            |3     |548.0    |3.2115214 |
+------+----------------------------------------------+------+---------+----------+
only showing top 10 rows
```

图 13-27 查看训练结果

思考：

为什么不把模型训练和 13.3.3 小节构建特征工程放入同一个 Pipeline 中？答案见本章“回顾思考”篇。

步骤 4：模型评估

在完成模型训练之后，我们需要了解模型的效果到底如何，需要通过量化的方式进行分析，我们使用 RegressionEvaluator 基于测试集检测模型的 RMSE 值，如图 13-28 所示。

```
In [89]: from pyspark.ml.evaluation import RegressionEvaluator

In [90]: evaluator = RegressionEvaluator(
             metricName='rmse',
             predictionCol='prediction',
             labelCol='rating')

In [91]: rmse = evaluator.evaluate(predicted_df)

In [92]: print('RMSE: ', rmse)

RMSE:  1.0146951683205292
```

图 13-28 查看模型 RMSE

通过结果可以看出，模型的 RMSE 值大概为 1.01，结果并不是非常高，这是因为实际评分和预测评分中存在错误，也就是我们通常所说的“脏数据”，不过，我们可以通过调整模型参数的方法来改进和优化。

13.3.8 推荐系统展示

在完成上述步骤，即模型训练与评估之后，我们需要将模型应用于我们的推荐系统，直观的向用

户展示推荐的电影，本小节我们将构建电影推荐系统。

步骤 1：获取电影集

我们需要根据数据集获取所有电影名，并将该 DataFrame 命名为 a，代码如下图 13-29 所示。

```
In  [99]: unique_movies = index_df.select('title_new').distinct()

In  [100]: unique_movies.count()

Out[100]: 1664

In  [103]: a = unique_movies.alias('a')

In  [106]: a.show(5)
+----------+
|title_new|
+----------+
|    558.0|
|    305.0|
|    299.0|
|    596.0|
|    769.0|
+----------+
only showing top 5 rows
```

图 13-29 获取电影名

由上图可知，我们的数据集中总共有 1664 部电影。

步骤 2：为用户推荐电影

我们选择为 user_id=85 的用户进行电影推荐，当然，非常重要的一点是，我们不能为用户推荐已经看过的电影，这样会非常影响用户体验，所以我们需要把用户看过或已进行评分的电影进行过滤，如下图 13-30 所示。

```
In  [107]: user_id = 85

In  [ ]: # creating another dataframe which contains already watched movie by active user

In  [109]: watched_movies = index_df\
               .filter(index_df['userId'] == user_id)\
               .select('title_new')\
               .distinct()

In  [110]: watched_movies.count()

Out[110]: 287
```

图 13-30 创建用户已看/评分电影

可以看到，1664 部电影中，用户已经看过或者评分过的电影有 287 部。我们将用户看过/评价过的电影数据 DataFrame 命名为 b。

步骤 3：合并与过滤

下面我们将 a 和 b 两张表进行关联合并，并从中过滤出可以推荐的电影，如下图 13-31 所示。

```
In [113]: total_movies = a.join(
              b,
              a.title_new == b.title_new,
              how='left')

In [114]: total_movies.show(5, False)
```

```
+----------+----------+
|title_new|title_new|
+----------+----------+
|558.0     |null      |
|305.0     |305.0     |
|299.0     |null      |
|596.0     |null      |
|769.0     |null      |
+----------+----------+
only showing top 5 rows
```

图 13-31 合并电影集

步骤 4：过滤掉用户已经看过或评分过的电影

这里我们需要两步操作：

首先，我们将用户已经看过的电影进行过滤，即在 total_movies 中，b.title_new 为空的数据，并对这些数据进行 distinct 去重，得到用户没有看过的电影 id 序列，如图 13-32 代码中“In[121]”部分；然后，添加上用户 id，即 user_id 列，如图 13-32 代码中“In[125]”部分。具体完整代码如图 13-32 所示。

批注 [雷6]: 需对图片进行讲解

```
In [121]: remaining_movies = total_movies\
              .where(F.col("b.title_new").isNull())\
              .select(a.title_new)\
              .distinct()

In [122]: remaining_movies.show(5, False)
```

```
+----------+
|title_new|
+----------+
|558.0     |
|299.0     |
|596.0     |
|769.0     |
|934.0     |
+----------+
only showing top 5 rows
```

```
In [125]: remaining_movies = remaining_movies.withColumn("userId", F.lit(int(user_id)))

In [126]: remaining_movies.show(10, False)
```

```
+----------+------+
|title_new|userId|
+----------+------+
|558.0     |85    |
|299.0     |85    |
|596.0     |85    |
|769.0     |85    |
|934.0     |85    |
|1051.0    |85    |
|692.0     |85    |
|810.0     |85    |
|720.0     |85    |
|782.0     |85    |
+----------+------+
only showing top 10 rows
```

图 13-32 过滤掉用户已经看过或评分过的电影

步骤 5：推荐 TopN 电影

接下来，我们可以利用上面的数据结果为剩余电影来做推荐，我们选择模型预测具有高评分的电影来进行推荐，即按照预测得分列“prediction”进行倒序排序，然后取出前 N 项，这里我们使用了 show() 方法，传入的参数为 5，代表取出的前 5 项，如下图 13-33 所示。

```
In [128]: recommendations = rec_model\
              .transform(remaining_movies)\
              .orderBy('prediction',ascending=False)

In [129]: recommendations.show(5,False)
```

```
+----------+------+----------+
|title_new|userId|prediction|
+----------+------+----------+
|1328.0    |85    |4.8966446 |
|1271.0    |85    |4.7829847 |
|1132.0    |85    |4.7179346 |
|288.0     |85    |4.656434  |
|1367.0    |85    |4.6416006 |
+----------+------+----------+
only showing top 5 rows
```

图 13-33 获取高评分电影推荐结果

步骤 6：关联电影名

最后一步，我们不能直接将上一步的结果推荐给用户，因为这是我们通过 StringIndex 编码后的特征字段，我们需要使用 IndexToString 函数来创建一个可以返回电影名的数据列，如下图 13-34 所示。

```
In [145]: movie_title = IndexToString(
              inputCol="title_new",
              outputCol="title",
              labels=pipelineModel.stages[0].labels)

          final_recommendations = movie_title.transform(recommendations)

In [146]: final_recommendations.show(10, False)
```

```
+----------+------+----------+-----------------------------------------------+
|title_new|userId|prediction|title                                          |
+----------+------+----------+-----------------------------------------------+
|1328.0    |85    |4.8966446 |Legal Deceit (1997)                            |
|1271.0    |85    |4.7829847 |Whole Wide World, The (1996)                   |
|1132.0    |85    |4.7179346 |Incognito (1997)                               |
|288.0     |85    |4.656434  |Hoop Dreams (1994)                             |
|1367.0    |85    |4.6416006 |Maya Lin: A Strong Clear Vision (1994)         |
|285.0     |85    |4.621625  |Wrong Trousers, The (1993)                     |
|514.0     |85    |4.5345607 |Jean de Florette (1986)                        |
|967.0     |85    |4.5279994 |Thirty-Two Short Films About Glenn Gould (1993)|
|1468.0    |85    |4.5212555 |Anna (1996)                                    |
|638.0     |85    |4.498848  |Shall We Dance? (1996)                         |
+----------+------+----------+-----------------------------------------------+
only showing top 10 rows
```

图 13-34 关联电影名称

从上述结果可以看出，模型为 user_id=85 的用户推荐的评分最高的电影为《Legal Deceit (1997)》。

步骤 7：封装推荐模型

在上面步骤其实已经完成了电影推荐系统的任务，但是为了更完整，我们将上面的模型推荐与处理进行封装，封装代码如下图 13-35 所示。

In [147]:
```
def get_top_movies(user_id, n):
    """
    This function returns the top 'n' movies that user has not seen yet but might like

    """
    #assigning alias name 'a' to unique movies df
    a = unique_movies.alias('a')

    #creating another dataframe which contains already watched movie by active user
    watched_movies = index_df\
        .filter(index_df['userId'] == user_id)\
        .select('title_new')\
        .distinct()

    #assigning alias name 'b' to watched movies df
    b = watched_movies.alias('b')

    #joining both tables on left join
    total_movies = a.join(
        b,
        a.title_new == b.title_new,
        how='left')

    #selecting movies which active user is yet to rate or watch
    remaining_movies = total_movies\
        .where(F.col("b.title_new").isNull())\
        .select(a.title_new)\
        .distinct()


    #adding new column of user_Id of active useer to remaining movies df
    remaining_movies = remaining_movies.withColumn("userId", F.lit(int(user_id)))


    #making recommendations using ALS recommender model and selecting only top 'n' movies
    recommendations = rec_model\
        .transform(remaining_movies)\
        .orderBy('prediction',ascending=False)\
        .limit(n)


    #adding columns of movie titles in recommendations
    movie_title = IndexToString(
        inputCol="title_new",
        outputCol="title",
        labels=pipelineModel.stages[0].labels)

    final_recommendations = movie_title.transform(recommendations)

    #return the recommendations to active user
    return final_recommendations.show(truncate=False)
```

图 13-35 封装电影推荐系统函数

接下来，我们再次对封装后的推荐系统函数进行验证与测试，如下图 13-36 所示。

In [149]:
```
get_top_movies(85, 10)
```

```
+---------+------+----------+-----------------------------------------------------+
|title_new|userId|prediction|title                                                |
+---------+------+----------+-----------------------------------------------------+
|1328.0   |85    |4.8966446 |Legal Deceit (1997)                                  |
|1271.0   |85    |4.7829847 |Whole Wide World, The (1996)                         |
|1132.0   |85    |4.7179346 |Incognito (1997)                                     |
|288.0    |85    |4.656434  |Hoop Dreams (1994)                                   |
|1367.0   |85    |4.6416006 |Maya Lin: A Strong Clear Vision (1994)               |
|285.0    |85    |4.621625  |Wrong Trousers, The (1993)                           |
|514.0    |85    |4.5345607 |Jean de Florette (1986)                              |
|967.0    |85    |4.5279994 |Thirty-Two Short Films About Glenn Gould (1993)      |
|1468.0   |85    |4.5212555 |Anna (1996)                                          |
|638.0    |85    |4.498848  |Shall We Dance? (1996)                               |
+---------+------+----------+-----------------------------------------------------+
```

图 13-36 推荐结果展示

可以看到，封装后的函数推荐的结果与上面步骤完全相同，从而也验证了我们封装后电影推荐系统

函数的正确性。至此，我们完整的实现了电影推荐的项目。

★回顾思考★

01　13.3.6 小节中的思考？

答：由于构建特征是对电影名编码为数字，而后拆分为训练集和测试集，如果将构建特征工程的 stage 和模型训练放入同一个 pipeline，则可能会报错，因为训练集中的电影名不一定完全包含测试集中的电影名。读者可以进行自行尝试。

02　推荐系统常用的方法有哪些类？

答：
（1） 基于内容的推荐系统
（2） 基于协同过滤的推荐系统
（3） 混合推荐系统

★练习★

一、选择题

1. ALS 模型属于哪一类推荐系统（ ）
   A. 基于内容的推荐系统
   B. 基于规则的推荐系统
   C. 基于协同过滤的推荐系统
   D. 混合推荐系统
2. orderBy()函数传入哪个函数可以实现随机打印（ ）
   A. random()
   B. rand()
   C. random(rand())
   D. randomint()
3. Pyspark DataFrame 使用哪个算子可以对数据进行去重（ ）
   A. unique()
   B. distint()
   C. distinct()
   D. set()

二、判断题

1. ALS 模型可以使用 RMSE 进行评估。（ ）
2. StringIndexer 方法不需要 fit()。（ ）

三、实战练习

任务 1：创建 Notebook 文件并完整实现本章节电影推荐系统项目。请参考本章配套 Notebook 代码文件。

批注 [雷7]: 需完成答案

批注 [HZ8R7]: 这个是一个代码文件。

本章小结

本章介绍了大数据分析处理中应用最为广泛的一个领域——推荐系统，它是大数据与机器学习的深度结合。本章首先对推荐系统的定义以及常见的推荐系统的方法进行了深入介绍，然后通过实战方式从读取数据集、数据集预处理、训练/评估模型、模型预测等多个方面完整的进行了代码展示。通过本章的学习，能够把本书所涉及的核心部分进行串联，使知识得到进一步的巩固。