

## 第 11 章 Spark 机器学习利器

### ★本章导读★

本章将介绍 Spark 机器学习库的基本使用方法。如果说 Spark 作为大数据分析处理银河中最闪亮的一颗星星，Spark 机器学习则是 Spark 中最明亮的核心部分。ML/Mlib 是构建在 Spark 之上，专门针对大数据分析处理的并发式高速机器学习库，其使得数据的计算处理速度大大高于普通的数据处理引擎。本章节将介绍 Mlib 与 ML 的区别，并围绕 ML 进行 Spark 机器学习的数据分析处理的学习。

### ★知识要点★

通过本章内容的学习，读者将掌握以下知识：

- Spark Mlib 基本数据类型
- Spark ML 和 Mlib 的联系与区别
- 使用 Spark ML 进行机器学习算法的开发

### 11.1 Spark ML 与 Spark Mlib

本小节我们将通过逻辑回归实战的方式完成 Spark ML Pipeline 的实现。

#### 11.1.1 Spark Mlib 介绍

MLlib 是 Spark 的机器学习（Machine Learning）库，旨在简化机器学习的工程实践工作，并方便扩展到更大规模。MLlib 由一些通用的学习算法和工具组成，包括分类、回归、聚类、协同过滤、降维等，同时还包括底层的优化原语和高层的管道 API。

Spark Mlib 包含基于 RDD 的原始算法 API。Spark MLlib 历史比较长，在 1.0 以前的版本就已经包含了，提供的算法实现都是基于原始的 RDD，如图 11-1 所示。

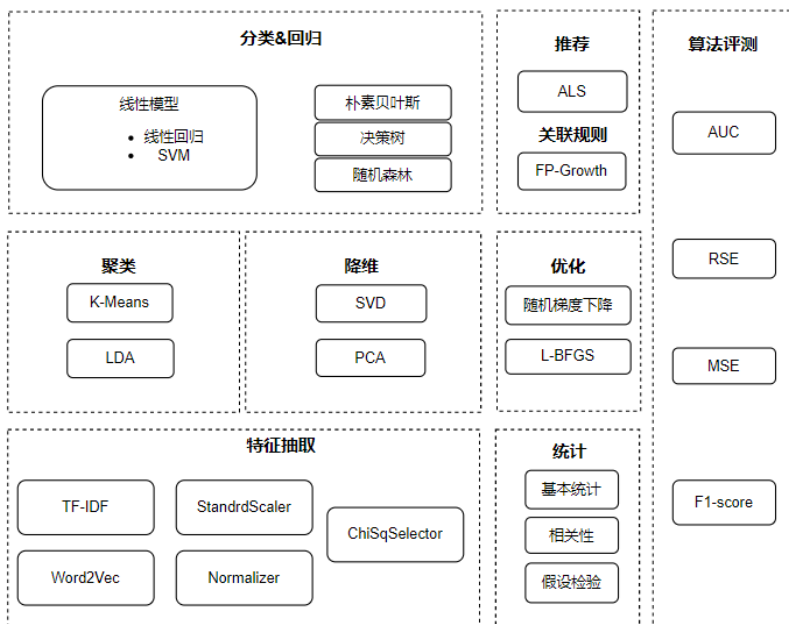


图 11-1 Spark Mlib 构成

Mlib 作为 Spark 的核心处理引擎。RDD 是 Mlib 专用的数据格式类型，它参考了 Scala 函数式编程的思想。Mlib 支持多种数据类型，Mlib 支持的数据类型如表 11-1 所示。

表 11-1 Mlib 支持的数据类型

名称	含义
<b>Local vector</b>	本地向量集
<b>Labeled point</b>	向量标签
<b>Local matrix</b>	本地矩阵
<b>Distributed matrix</b>	分布式矩阵

### 11.1.2 Spark ML 介绍

正如前面小节所介绍，Spark Mlib 所提供的算法实现都是基于原始的 RDD。Spark ML 则提供了基于 DataFrames 高层次的 API，可以用来构建机器学习工作流(PipeLine)。ML Pipeline 弥补了原始 MLib 库的不足，向用户提供了一个基于 DataFrame 的机器学习工作流式 API 套件。

使用 ML Pipeline API 可以很方便的把数据处理，特征转换，正则化，以及多个机器学习算法联合起来，构建一个单一完整的机器学习流水线。这种方式给我们提供了更灵活的方法，更符合机器学习过程的特点，也更容易从其他语言迁移。Spark 官方推荐使用 spark.ml。如果新的算法能够适用于机器学习管道的概念，就应该将其放到 spark.ml 包中，如：特征提取器和转换器。开发者需要注意的是，从 Spark2.0 开始，基于 RDD 的 API 进入维护模式（即不增加任何新的特性），并预期于 3.0 版本的时候被移除除 MLLib。

### 11.1.3 Spark ML 与 Spark Mlib 区别

Spark ML 与 Spark Mlib 的主要区别和联系有如下几个方面：

1. ML 和 Mlib 都是 Spark 中的机器学习库，目前常用的机器学习功能 2 个库都能满足需求。
2. ML 是升级版的 MLlib，最新的 Spark 版本优先支持 ML。
3. 二者面向的数据集不一样，Mlib 操作的是 RDD，而 ML 主要操作的是 DataFrame。
4. 相比于 Mlib 在 RDD 提供的基础操作，ML 在 DataFrame 上的抽象级别更高，数据和操作耦合度更低。
5. ML 支持 Pipelines，可以把很多操作（算法/特征提取/特征转换）以管道的形式串起来。
6. ML 中的随机森林支持更多的功能：包括重要度、预测概率输出等，而 MLlib 不支持。

Spark 官方推荐使用 ML，因为 ML 功能更全面更灵活，同样 ML 也是 Spark 机器学习的重大趋势，是大数据分析处理的一大利器。

## 11.2 Spark Mlib 数据类型

### 11.2.1 本地向量集

Mlib 使用的本地化存储类型是向量，向量又包含两种形式：密集向量数据集（dense）和稀疏向量数据集（spares）。本地向量的使用需要引入如下库。

```
from pyspark.mllib.linalg import Vectors
```

密集向量数据集和稀疏向量数据集创建的代码分别如如图 11-2、图 11-3 所示。

```
In [63]: vd = Vectors.dense(2, 0, 3)
```

```
In [64]: vd
```

```
Out[64]: DenseVector([2.0, 0.0, 3.0])
```

图 11-2 密集向量数据集创建

```
In [65]: vd = Vectors.sparse(4, [1, 3], [2, 3])
```

```
In [66]: vd
```

```
Out[66]: SparseVector(4, {1: 2.0, 3: 3.0})
```

图 11-3 稀疏向量数据集创建

从上图可以看出，同一个向量数据，密集向量数据集是将数据集作为一个集合的形式整体进行存储，和 Array 格式类似。而稀疏向量数据集将向量的存储变成 4 个部分，第一部分，即参数 4，代表输入数据的大小；第二个参数是数据下标的数值；第三个参数是值输入的数据。值得注意的是，Mlib 的数据支持格式中仅支持整数型与浮点型数据。

### 11.2.2 向量标签

向量标签用于对 Mlib 机器学习算法的不同值做标记。例如在分类问题中，可以将不同的数据集进行分组，分成若干份，以整数 0、1、2、... 进行标记，最简单的例如二分类问题中的 label。

LabeledPoint 是建立向量标签的静态类，主要有两个方法，Features 用于显示打印标记点所代表的数据内容，而 Label 用于显示标记数。

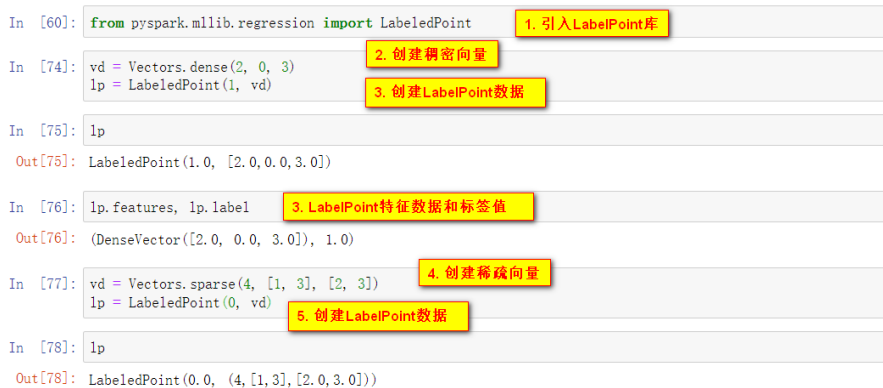
除了使用直接创建方法建立向量标签之外，Mlib 还支持从数据库或者文件中直接读取数据集的方式创建，当然，所读取的数据集需要有指定格式，具体格式如下：

```
label key1:value1 key2:value2 key3:value3 ...
```

接下来，我们通过代码创建向量标签，创建向量标签需要引入如下库。

```
from pyspark.mllib.regression import LabeledPoint
```

具体实现代码如下图 11-4 所示。



```
In [60]: from pyspark.mllib.regression import LabeledPoint
Out[60]: LabeledPoint(1.0, [2.0, 0.0, 3.0])

In [74]: vd = Vectors.dense(2, 0, 3)
         lp = LabeledPoint(1, vd)
Out[74]: LabeledPoint(1.0, [2.0, 0.0, 3.0])

In [75]: lp
Out[75]: LabeledPoint(1.0, [2.0, 0.0, 3.0])

In [76]: lp.features, lp.label
Out[76]: (DenseVector([2.0, 0.0, 3.0]), 1.0)

In [77]: vd = Vectors.sparse(4, [1, 3], [2, 3])
         lp = LabeledPoint(0, vd)
Out[77]: LabeledPoint(0.0, [4.0, 1.3, 2.0, 3.0])

In [78]: lp
Out[78]: LabeledPoint(0.0, [4.0, 1.3, 2.0, 3.0])
```

图 11-4 创建 LabeledPoint 格式数据

### 11.2.3 本地矩阵

在面对大数据分析处理时，普通格式数据的计算往往效率较低，为了解决大数据运算处理，提高计算效率，Mlib 加入了矩阵的概念，矩阵又包含本地矩阵和分布式矩阵，部署在单机环境中的存储方法就是本地矩阵。一般来说，采用分布式矩阵进行存储的情况都是数据非常庞大的，单机远远不能满足数据处理速度和效率的要求，分布式矩阵的处理速度和效率与其存储格式是息息相关的，Mlib 提供了四种分布式矩阵的存储形式，均由整型、浮点型数据内容构成，这四种矩阵分别是：行矩阵、带有行索引的行矩阵、坐标矩阵和块矩阵，其中行矩阵和本地矩阵形式相同，由于一般不使用上述格式的分布式矩阵，所以本书仅对分布式矩阵做简单介绍，本章节只对本地矩阵展开讲解。

我们将数组数据创建成一个 3 行 2 列的本地矩阵，具体包含两个步骤：首先，我们需要引入 Matrices 矩阵库；然后，通过该库中的 dense 函数来创建本地矩阵。实现代码如下图 11-5 所示。

```
In [80]: from pyspark.mllib.linalg import Matrices
In [88]: data = [1,2,3,4,5,6]
         mx = Matrices.dense(2, 3, data)
In [89]: mx
Out[89]: DenseMatrix(2, 3, [1.0, 2.0, 3.0, 4.0, 5.0, 6.0], False)
```

1. 引入矩阵库

2. 创建本地矩阵

批注 [雷1]: 图片步骤内容需在正文中描述出来

图 11-5 本地矩阵创建

从上图代码可以看出，原始数据被拆分成了一个 3 行 2 列的矩阵。

值得注意的是，我们可以将本地矩阵转换成 `numpy.array()` 格式的数组，方便我们进行数据分析处理，实现代码如下图 11-6 所示。

```
In [93]: mx_arr = mx.toArray()
         print(mx_arr, type(mx_arr))

[[1.  3.  5.]
 [2.  4.  6.]] <class 'numpy.ndarray'>
```

图 11-6 本地矩阵转 Numpy

通过 `toArray()` 的方法可以直接将本地矩阵进行转换。

### 11.3 Spark Mlib 数据统计类型

本小节将介绍 Mlib 的数据统计相关内容，数据分析处理往往与数理统计是紧密关联的，数理统计是随着概率论发展起来的一个数学分支。Mlib 提供了常用的基本的数理统计方法，可以极大的方便用户进行数据分析与处理计算。目前，Mlib 仅包含以下基本的统计方法。

#### 11.3.1 统计量基本数据

首先，在数理统计中，我们要了解基本的统计量，如平均值、方差、标准差等等。Mlib 中的统计量基本数据如表 11-2 所示。

表 11-2 Mlib 中统计量基本数据

名称	含义
count	行内个数统计
Max	最大数值（列方向比较）
Min	最小数值（列方向比较）
normL1	曼哈顿距离
normL2	欧几里得距离
numNonzeros	不包含 0 值的个数
variance	标准差（列方向比较）

接下来，我们通过代码的方式实现数据基本统计量的计算，我们首先创建一个 3 行 3 列数据的 RDD，然后对该 RDD 创建统计实例，如图 11-6 中 “In[111]” 和 “In[112]” 部分代码所示，我们分别对该数据

计算最大值、最小值、平均值和标准差，具体代码如图 11-6 所示。。

```
In [111]: rdd = sc.parallelize([Vectors.dense([1, 2, 4]),  
                             Vectors.dense([2, 2, 2]),  
                             Vectors.dense([2, 2, 3]),  
                             ])
```

1. 创建3行向量

```
In [112]: stat = Statistics.colStats(rdd)
```

2. 获取Statistics实例

```
In [113]: stat.max()  
Out[113]: array([2., 2., 4.])
```

3. 计算最大值，此时为按行比较，即列方向的最大值，例如第3列最大值在第一行，值为4

```
In [114]: stat.min()  
Out[114]: array([1., 2., 2.])
```

4. 计算最小值

```
In [115]: stat.mean()  
Out[115]: array([1.66666667, 2., 3.])
```

5. 计算平均值

```
In [116]: stat.variance()  
Out[116]: array([0.33333333, 0., 1.])
```

6. 计算标准差

图 11-6 基本统计量的计算

批注 [雷2]: 图片内容需要正文中描述出来，全文相同问题同样处理

11.3.2 相关系数计算

在数据分析处理中，我们通常需要衡量两组数据之间的相关关系，通常是指线性相关关系，那么反映两变量线性相关关系的统计指标称为相关系数。相关系数是一种用来反映变量之间相关关系程度的一种统计指标，一般用于对两组数据的拟合、相似程度进行定量化分析。常见的相关系数一般有皮尔逊系数和斯皮尔曼系数，后者相对使用的较少，但是能够较好地反应不同数据集的趋势程度，因此在实际应用汇总中也是有所涉及的，本小节只对皮尔逊系数进行介绍，感兴趣的读者可以查阅皮尔逊系数相关资料。

皮尔逊相关系数计算公式如下：

$$\rho_{xy} = \frac{\sum (x - \bar{x})(y - \bar{y})}{\sqrt{\sum (x - \bar{x})^2 \sum (y - \bar{y})^2}}$$

计算皮尔逊相关系数代码如图 11-7 所示。

```
In [122]: x = sc.parallelize([1.0, 0.0, -2.0])  
         y = sc.parallelize([4.0, 5.0, 3.0])  
  
In [123]: Statistics.corr(x, y, "spearman")
```

使用 spearman 方法

```
Out[123]: 0.5
```

图 11-7 皮尔逊相关系数代码

温馨提示：

皮尔逊相关系数代表两组数据的余弦分开程度，表示随着数据量的增加，两组数据差别将增大。

### 11.3.3 卡方检验计算

上一小节介绍的相关系数计算方法一定程度上可以表征数据集之前的相关程度，但是对于数据结果的质量，同样需要一个能够定量反应和检验结果是否正确的方法。

常见的检验方法有卡方检验、t 检验等等。卡方检验是常用的一种假设检验方法，能够较好地反应数据集之间的拟合程度、相关性以及独立性。Mlib 规定常用的卡方检验使用的数据集一般为向量或者矩阵。

卡方假设检验实现代码如图 11-8 所示。

```
In [133]: vd = Vectors.dense([1,2,3,4,5,6])
In [136]: cd_chi = Statistics.chiSqTest(vd)
In [138]: print(cd_chi)

Chi squared test summary:
method: pearson
degrees of freedom = 5
statistic = 5.0000000000000001
pValue = 0.4158801869955079
No presumption against null hypothesis: observed follows the same distribution as expected..
```

图 11-8 稠密向量卡方检验

从上图结果可以看出，假设检验的输出结果包含三部分，分别是自由度、P-value 以及统计量，这些统计量的具体含义如表 11-3 所示。

表 11-3 假设检验常用统计量介绍

名称	含义
自由度	总体参数估计量中变量值独立自由变化的数目
P-value	显著性差异指标
方法	卡方检验使用方法

一般来说，P-value 小于 0.05 时，指数据集不存在显著性差异。

当然，我们还可以对本地矩阵进行卡方检验，代码如下图 11-9 所示。

```
In [139]: mx = Matrices.dense(2, 3, [1,2,3,4,5,6])
          cd_chi = Statistics.chiSqTest(mx)
          print(cd_chi)

Chi squared test summary:
method: pearson
degrees of freedom = 2
statistic = 0.14141414141414146
pValue = 0.931734784568187
No presumption against null hypothesis: the occurrence of the outcomes is statistically independent..
```

图 11-9 本地矩阵卡方检验

## 11.4 线性回归

### 11.4.1 线性回归介绍

线性回归是机器学习中经常使用，并且最简单的一个机器学习算法。回归分析是一种预测性建模技术，主要用于研究因变量和自变量之间的关系，通常被用于预测分析、时间序列等。

简单来说，回归分析就是使用曲线（直线是特殊的曲线）来拟合某些已知的数据点，使数据点离曲线或曲面的距离达到最小。自变量和因变量之间的关系主要有两种：

1. 线性关系
2. 非线性关系

任意两个变量之间存在线性关系，那么就可以使用回归分析的方法进行建模和预测，这便是线性回归。线性回归的总体目标是预测一条贯穿数据的直线，使得每个数据点到这条直线的垂直距离都是最小的。线性回归是利用线性回归方程的最小二乘法对一个或多个自变量和因变量之间关系进行建模的方法。

### 11.4.2 最小二乘法

本小节将详细介绍线性回归的基本思路和最常使用的求解方法——最小二乘法。

假设可以找到最佳拟合的直线方程： $y = ax + b$ ，如下图 11-10 所示。

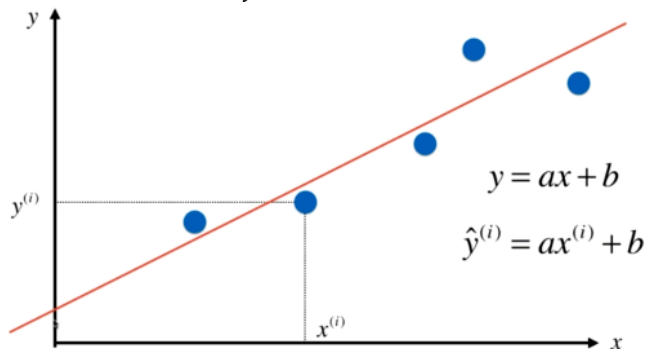


图 11-10 线性回归示意图

线性回归模型的目标就是要求解参数  $a$  和  $b$ ，当然，上图是在二维平面中的线性回归，即一元线性回归，在实际中，线性回归可能指的是多元线性回归， $a$  和  $b$  为向量，即维度大于等于 1，即：

$$y = (y_1, y_2, \dots, y_n)$$

$$x = (x_1, x_2, \dots, x_n)$$

则对于每个样本点  $x^{(i)}$ ，根据得到的直线方程，可以得到预测值为： $\hat{y}^{(i)} = ax + b$ 。真值为  $y^{(i)}$ ，希望  $\hat{y}^{(i)}$  和  $y^{(i)}$  的差距尽可能的小，表达式  $\hat{y}^{(i)}$  和  $y^{(i)}$  的差距为  $y^{(i)} - \hat{y}^{(i)}$ 。考虑到所有样本，即误差可能出现正负误差相抵的情况，所以采用误差平方和，如下所示。

$$\sum_{i=1}^m (y^{(i)} - \hat{y}^{(i)})^2$$



我们的目标是使上式尽可能的小，结合预测直线方程，即：

$$\sum_{i=1}^m (y^{(i)} - ax^{(i)} - b)^2$$

尽可能小，该式也叫损失函数（Loss Function）。求最大值的叫效用函数（Utility Function）

对于回归问题，通常采用的策略是使用最小均方差损失来判断模型的好坏，均方误差也叫做平方损失。最小误差的平方是典型的最小二乘法问题。

求

$$J(a,b) = \sum_{i=1}^m (y^{(i)} - ax^{(i)} - b)^2$$

最小值，即基于均方误差最小化来进行模型求解的方法叫做“最小二乘法”。在线性回归中，最小二乘法就是试图找到一条直线，使得所有样本到直线上的欧氏距离之后最小。

上述求解使得最小化的过程，称为线性回归模型的最小二乘“参数估计”。

最小二乘法应用范围很广，不仅限于线性回归。在线性回归中，是关于  $J(a,b)$  的凸函数，当关于  $a$  和  $b$  的导数均为 0 时，得到  $a$  和  $b$  的最优解。

这里对凸函数的定义进行简单的说明，凸函数，即在实数集上的函数，可以通过求二阶导数来判断：

1. 如果二阶导数在区间上非负，则称为凸函数，恒大于 0，则称为严格凸函数；
2. 如果二阶导数在区间上非正，则称为凹函数，恒小于 0，则称为严格凹函数。

采用链式求导法则：

$$\begin{cases} \frac{\partial J(a,b)}{\partial a} = 0 \\ \frac{\partial J(a,b)}{\partial b} = 0 \end{cases}$$

其中：

$$\begin{aligned} \frac{\partial J(a,b)}{\partial a} &= 2 \sum_{i=1}^m (y^{(i)} - ax^{(i)} - b)(-x^{(i)}) = 0 \\ &= \sum_{i=1}^m (y^{(i)} - ax^{(i)} - \bar{y} + a\bar{x})x^{(i)} \\ &= \sum_{i=1}^m (x^{(i)}y^{(i)} - x^{(i)}\bar{y}) - \sum_{i=1}^m (ax^{(i)^2} - a\bar{x}x^{(i)}) \\ &= \sum_{i=1}^m (x^{(i)}y^{(i)} - x^{(i)}\bar{y}) - a \sum_{i=1}^m ((x^{(i)})^2 - \bar{x}x^{(i)}) = 0 \\ \frac{\partial J(a,b)}{\partial b} &= 2(m\bar{b} - \sum_{i=1}^m (y^{(i)} - ax^{(i)})) = 0 \end{aligned}$$

最终：

$$a = \frac{\sum_{i=1}^m (x^{(i)} - \bar{x})(y^{(i)} - \bar{y})}{\sum_{i=1}^m (x^{(i)} - \bar{x})^2}$$

$$b = \bar{y} - a\bar{x}$$

### 11.4.3 创建 SparkSession

从本小节开始，正式进入 Pyspark ML 机器学习的实战环节。首先新建本章节的 Jupyter Notebook 文件。

步骤 1: 新建 Notebook

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 11-11 所示。

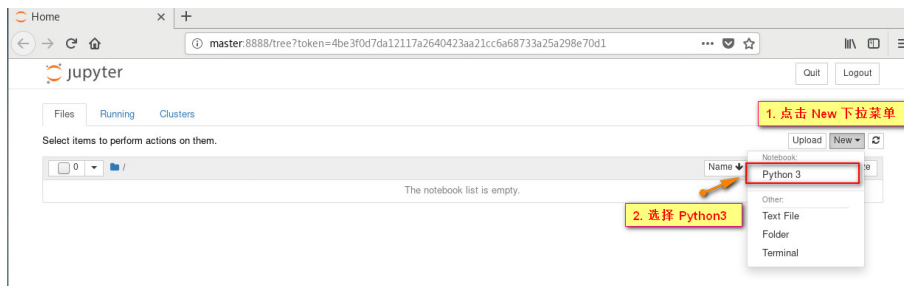


图 11-11 新建 Notebook 文件

步骤 2: 重命名 Notebook

单击【Untitled】，输入“Chapter11”，并确认，为 Notebook 重命名，如图 11-12 所示。

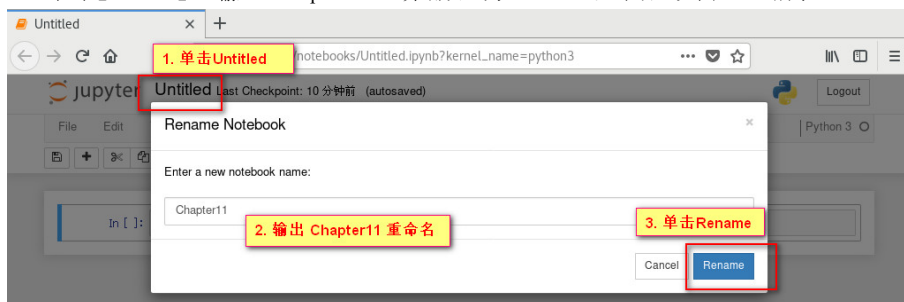


图 11-12 重命名 Notebook 文件

步骤 3: 创建 SparkSession

在完成上述步骤之后，在 Notebook 文件中输入如下代码，创建本小节线性回归实例的 SparkSession，如下图 11-13 所示。

```
In [141]: from pyspark.sql import SparkSession
spark = SparkSession\
    .builder\
    .appName('line_reg')\
    .getOrCreate()
```

```
In [142]: spark
```

```
Out[142]: SparkSession - hive
SparkContext
```

[Spark UI](#)

**Version**

v2.4.3

**Master**

local[\*]

**AppName**

line\_reg

图 11-13 创建线性回归 SparkSession

在完成创建 SparkSession 后，我们需要获取当前代码的工作目录，已便于读取数据集（为便于读者快速上手，这里使用本地数据集，Pyspark Notebook 为 Local 模式。当然可以启动集群模式，使用 HDFS 文件），如图 11-14 所示。

```
In [1]: pwd = !pwd
data_path = "file://" + list(pwd)[0] + '/data/ai/'
print(data_path)

file:///root/pyspark-book/data/ai/
```

图 11-14 获取工作路径

#### 11.4.4 读取数据集

线性回归使用的数据集文件为 linear\_regression.csv。下面我们将读取数据集，并对数据集进行探索。

步骤 1: 读取数据集

代码如图 11-15 所示。

```
In [4]: df = spark.read.csv(data_path + 'linear_regression.csv', header=True, inferSchema=True)
```

```
In [5]: df.show(3)
```

var_1	var_2	var_3	var_4	var_5	output
734	688	81	0.328	0.259	0.418
700	600	94	0.32	0.247	0.389
712	705	93	0.311	0.247	0.417

only showing top 3 rows

图 11-15 读取数据集

步骤 2: 查看数据结构

代码如图 11-16 所示。

```
In [7]: df.printSchema()

root
|-- var_1: integer (nullable = true)
|-- var_2: integer (nullable = true)
|-- var_3: integer (nullable = true)
|-- var_4: double (nullable = true)
|-- var_5: double (nullable = true)
|-- output: double (nullable = true)
```

图 11-16 查看数据结构

从上图可以看出，数据集各字段类型为 `integer` 或 `double`，这是由于我们在步骤 1 读取数据的时候通过参数 `inferSchema` 指定了按 `csv` 文件字段数据类型读取，否则读取字段可能全为 `string` 类型，导致无法使用线性回归，读者可以自行尝试。

步骤 3: 查看相关系数

我们可以使用 `pyspark` 自带的函数查看某列特征与输出值之间的相关系数，代码如下图 11-17 所示。

```
In [9]: import pyspark.sql.functions as F

In [10]: df.select(F.corr('var_1', 'output')).show()

+-----+
|corr(var_1, output)|
+-----+
| 0.9187399607627283|
+-----+
```

图 11-17 查看相关系数

#### 11.4.5 特征工程

在简单探索数据内容和结构之后，我们将对数据进行预处理和特征工程构建，这里我们使用 `VectorAssembler` 方法，将所有输入特征合成一个向量特征。

步骤 1: 创建 `VectorAssembler`

`VectorAssembler` 有两个参数：

- `inputCols`: 需要进行合并的列名（List 类型）
- `outputCol`: 合并后输出的列名（字符串类型）

代码如图 11-18 所示。

```
In [12]: from pyspark.ml.linalg import Vector
         from pyspark.ml.feature import VectorAssembler

In [13]: vec_assembler = VectorAssembler(
         inputCols=['var_1', 'var_2', 'var_3', 'var_4', 'var_5'],
         outputCol='features'
         )
```

图 11-18 创建 `VectorAssembler`

温馨提示:

本章节侧重于常用机器学习方法的原理以及如何使用 Spark 来构建机器学习模型, 其中涉及了如 `VectorAssembler` 方法、`StringIndexer` 方法、`OneHotEncoder` 方法以及对模型 `fit` 和 `transform` 的操作, 读者仅需掌握使用即可, 这些方法的具体内容会在下一章节中详细站内介绍和练习。

### 步骤 2: 转换原始数据集

这里, 使用 `transform()` 方法进行转换, 可以理解为通过 `transform` 之后即完成将上述步骤创建的合并多列为一列的处理转换, 具体代码如图 11-19 所示。

```
In [14]: features_df = vec_assembler.transform(df)

In [15]: features_df.show(3, False)

+-----+-----+-----+-----+-----+-----+
|var_1|var_2|var_3|var_4|var_5|output|features|
+-----+-----+-----+-----+-----+-----+
|734  |688  |81   |0.328|0.259|0.418|[734.0,688.0,81.0,0.328,0.259]|
|700  |600  |94   |0.32  |0.247|0.389|[700.0,600.0,94.0,0.32,0.247]|
|712  |705  |93   |0.311|0.247|0.417|[712.0,705.0,93.0,0.311,0.247]|
+-----+-----+-----+-----+-----+-----+
only showing top 3 rows

In [16]: features_df.printSchema()

root
 |-- var_1: integer (nullable = true)
 |-- var_2: integer (nullable = true)
 |-- var_3: integer (nullable = true)
 |-- var_4: double (nullable = true)
 |-- var_5: double (nullable = true)
 |-- output: double (nullable = true)
 |-- features: vector (nullable = true)
```

图 11-19 转换原始数据集

### 步骤 3: 获取训练数据

由于我们使用 `VectorAssembler` 将所有特征合成了一个向量特征, 所以后续模型训练相关操作我们需要合成后的特征字段即可, 如下图 11-20 所示。

```
In [17]: model_df = features_df.select('features', 'output')

In [18]: model_df.show(5, False)

+-----+-----+
|features|output|
+-----+-----+
|[734.0,688.0,81.0,0.328,0.259]|0.418|
|[700.0,600.0,94.0,0.32,0.247]|0.389|
|[712.0,705.0,93.0,0.311,0.247]|0.417|
|[734.0,806.0,69.0,0.315,0.26]|0.415|
|[613.0,759.0,61.0,0.302,0.24]|0.378|
+-----+-----+
only showing top 5 rows
```

图 11-20 筛选数据集字段

### 11.4.6 构建训练集和测试集

在前面小节中，我们完成了数据集特征工程的构建，接下来我们将对数据集进行切分和模型的训练工作。

步骤 1：切分数据集

我们按照 7:3 的比例将数据集切分为训练集和测试集，如下图 11-21 所示。

```
In [20]: train_df, test_df = model_df.randomSplit([0.7, 0.3])

In [21]: print((train_df.count(), len(train_df.columns)))
(863, 2)

In [22]: print((test_df.count(), len(test_df.columns)))
(369, 2)
```

图 11-21 切分数据集

步骤 2：查看训练集描述

我们使用 `describe()` 方法查看训练集的数据量、平均数等维度信息，如下图 11-22 所示。

```
In [23]: train_df.describe().show()
```

summary	output
count	863
mean	0.39804866743916506
stddev	0.03266542685059044
min	0.301
max	0.485

图 11-22 查看训练集描述

### 11.4.7 模型训练

接下来，我们使用 `pyspark.ml` 库中的线性回归算法对训练集进行训练，构建线性回归模型，如下图 11-23 所示。

```
In [24]: from pyspark.ml.regression import LinearRegression

In [25]: model = LinearRegression(labelCol='output')

In [26]: line_model = model.fit(train_df)

In [28]: print("Line intercept:", line_model.intercept)
print("Line coefficients:\n", line_model.coefficients)

Line intercept: 0.19629213927216993
Line coefficients:
[0.00034834707065082687, 4.529439013419945e-05, 0.0001405600894934997, -0.6404804066457538, 0.45320821437302533]
```

图 11-23 训练线性回归模型

从上图代码输出中可以看出以下信息，如线性回归模型的截距以及各维度特征的系数值。

### 11.4.8 模型评估

线性回归有多种评估模型好坏的方法，例如均方根误差、平均绝对值误差等等。本小节将对常见的方法进行介绍，并对前面小节训练的模型进行评估。

#### 1. $R^2$

$R^2$ 的计算方式如下：

$$R^2 = 1 - \frac{SS_{residual}}{SS_{total}} = 1 - \frac{\sum_i (\hat{y}^{(i)} - y^{(i)})^2}{\sum_i (\bar{y} - y^{(i)})^2}$$

其中，分子代表残差平方和，表示使用我们的模型预测产生的误差；分母：总离差平方和，使用  $y = \bar{y}$  预测产生的误差。其中，分母也叫 **Baseline Model**，因为这里不考虑  $x$ ，直接预测  $\bar{y}$  和样本数  $y^{(i)}$  之间的误差，这样会产生很多的误差错误，而分子考虑了  $x$  对模型预测结果的作用，所以误差应该是会偏小的，会减少一些只考虑  $y$  产生的误差，同时也会产生一些由  $x$  引入的错误。所以通过公式计算可以得到模型拟合住的那些数据，所以公式最终求解得到的是模型没有产生错误的指标。

所以由上式可以得到以下结论：

- (1) 越大越好，当预测的模型没有产生任务错误时，得到的最大值是 1；
- (2) 当模型等于基准模型（Baseline Module）的时候， $R^2 = 0$ ；
- (3)  $R^2 < 0$ ，说明模型还不如基准模型（即训练的模型还不如不训练）。此时，很可能数据不存在任何线性关系。

#### 2. 均方误差 MSE

MSE(Mean Squared Error)，计算公式如下：

$$MSE = \frac{1}{m} \sum_{i=1}^m (y^{(i)} - \hat{y}^{(i)})^2$$

此时，存在一个问题，即量纲。比如  $y$  代表房价（万元），均方误差的结果代表万元的平方，所以需要和  $y$  统一量纲，这便引入了下面的均方根误差方法。

#### 3. 均方根误差 RMSE

RMSE(Root Mean Squared Error)，计算公式如下：

$$\sqrt{\frac{1}{m} \sum_{i=1}^m (y^{(i)} - \hat{y}^{(i)})^2} = \sqrt{MSE}$$

下面，我们将对测试集使用上述评估方法对线性回归模型的效果进行评估。

##### 步骤 1：测试集预测

在测试集上的预测非常简单，只需要调用模型的 `evaluate()` 方法并传入测试数据集即可。代码如下图 11-24 所示。

```
In [33]: predict = line_model.evaluate(test_df)
```

图 11-24 线性回归模型预测

##### 步骤 2：查看评估指标

我们分别使用上述评估指标进行模型的检验，如下图 11-25 所示。

```
In [36]: print("R^2:", predict.r2)
R^2: 0.8820258243690249
```

```
In [37]: print("RMSE:", predict.rootMeanSquaredError)
RMSE: 0.011875565934291885
```

```
In [38]: print("MSE:", predict.meanSquaredError)
MSE: 0.00014102906625971387
```

图 11-25 查看模型评估指标

从上图可以看出，线性回归模型在测试集上的 $R^2$ 值约为 0.88，RMSE 的值约为 0.1188，MSE 的值约为 0.0001，根据本小节评估指标的介绍，可以得出结论，我们训练的线性回归模型效果较为理想。

## 11.5 逻辑回归

### 11.5.1 逻辑回归介绍

逻辑回归既可以看做是回归算法，也可以看做是分类算法。通常逻辑回归被作为分类算法使用，值得注意的是，逻辑回归只可以用于解决二分类问题。逻辑回归模型就是将线性回归的结果输入到一个 sigmoid 函数，sigmoid 函数是一种非线性函数，将线性回归的连续值映射到 0~1，进而输出为类别“0”和类别“1”的概率，如下图 11-26 所示。

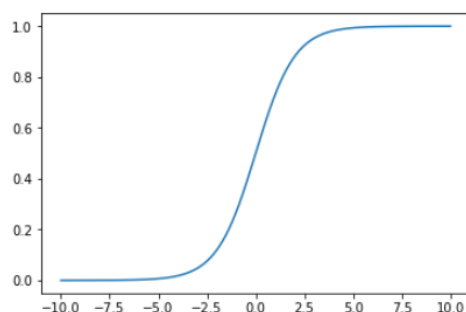


图 11-26 逻辑回归示意图

具体来看，线性回归的表达式为：

$$y_i = ax_i + b$$

上式中， $x_i$  是第  $i$  个样本的  $N$  个特征组成的特征向量，即  $x_i = (x_i^{(1)}, x_i^{(2)}, \dots, x_i^{(N)})$ ； $a$  为  $N$  个特征对应的特征权重组成的向量，即  $a = (a_i^{(1)}, a_i^{(2)}, \dots, a_i^{(N)})$ ； $b$  是第  $i$  个样本对应的偏置常数（截距）。



sigmoid 函数：

$$\delta_y = \frac{1}{1 + e^{-y}}$$

其中， $y$  是自变量， $\delta_y$  是因变量， $e$  是自然常数。

sigmoid 函数的图像如下图 11-27 所示。

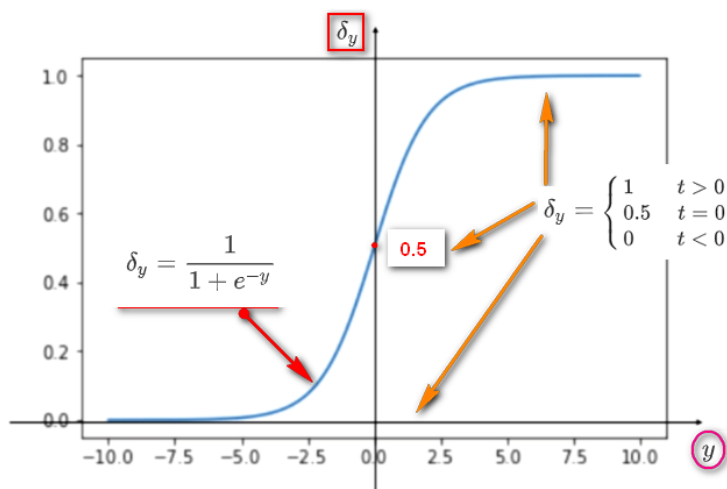


图 11-27 sigmoid 函数图像

在线性回归的输出值  $y$  的基础上再通过 sigmoid 函数，就可以得到逻辑回归的结果，为书写方便，我们令  $z_i = \delta_{y_i}$ ，即：

$$z_i = \delta_{y_i} = \frac{1}{1 + e^{-y_i}} = \frac{1}{1 + e^{-(a \cdot x_i + b)}}$$

如果我们将  $z_i = 1$  作为  $x_i$  为正样本的可能性，即：

$$P(z_i = 1 | x_i) = \frac{1}{1 + e^{-(a \cdot x_i + b)}} = \frac{e^{a \cdot x_i + b}}{1 + e^{a \cdot x_i + b}}$$

同理， $z_i = 0$  作为  $x_i$  为负样本的可能性为：

$$P(z_i = 0 | x_i) = 1 - P(z_i = 1 | x_i) = \frac{1}{1 + e^{a \cdot x_i + b}}$$

我们定义两者的比值为“概率”，即：

$$p = \frac{P(z_i = 1 | x_i)}{P(z_i = 0 | x_i)} = \frac{P(z_i = 1 | x_i)}{1 - P(z_i = 1 | x_i)}$$

我们对上式取对数，可得：

$$\ln p = \ln \frac{P(z_i = 1 | x_i)}{1 - P(z_i = 1 | x_i)} = a \cdot x_i + b$$

由上公式可得，对数概率  $\ln p$  的结果正好是线性回归的结果  $a \cdot x_i + b$ 。

由此可知，逻辑回归的本质其实就是用线性回归的预测结果去逼近真实标记的对数概率

$\ln \frac{y}{1-y}$ ，这也就是逻辑回归又被称为“对数概率回归”的原因。

### 11.5.2 逻辑回归模型评估

对于逻辑回归模型的评估，我们可以使用多个指标进行评判。对于二分类模型除了通过准确率之外还有其他指标，这就是混淆矩阵。

在分类任务下，预测结果(Predicted Condition)与正确标记(True Condition)之间存在四种不同的组合，构成混淆矩阵(适用于多分类)。

表 11-4 混淆矩阵

	正例	假例
正例	真阳性，正确的正例预测 (TP)	假阴性，错误的反例预测 (FN)
假例	假阳性，错误的正例预测 (FP)	真阴性，正确的反例预测 (TN)

下面我们详细说明混淆矩阵中每个元素的含义。

通常我们设定，正例为 1，反例为 0。那么，真阳性，正确的正例预测 (TP)，即指预测结果和真实值均为 1；假阳性，错误的正例预测 (FP)，即预测结果为 1，真实值为 0；假阴性，错误的反例预测 (FN)，即预测结果为 0，真实值为 1；真阴性，正确的反例预测 (TN)，即预测结果和真实值均为 0。

在有了混淆矩阵之后，我们通过判断模型正确性和性能的评判指标就可以很容易得出，常用指标有准确率、召回率、F1-score。

#### 1. 精准率

**精准率**，即预测结果为正例样本中真实为正例的比例。比如预测 10 个人为真，结果真实值为 8 个人真，2 个人为假，那么准确率为 0.8。对应混淆矩阵的示意图如下图 11-28 所示。

批注 [雷3]: 是准确率还是精确率？需修改统一

		预测结果	
		正例	假例
真实结果	正例	真正例TP	伪反例FN
	假例	伪正例FP	真反例TN

图 11-28 精准率示意图

精准率的计算公式为：

$$\frac{TP}{TP + FP}$$

## 2. 准确率

准确率，即预测正确的占总数的比例，其中预测正确包括正例和反例。通常，准确率并不是首选的指标，因为目标类别是不均衡的，对应混淆矩阵的示意图如下 11-29 所示。

		预测结果	
		正例	假例
真实结果	正例	真正例TP	伪反例FN
	假例	伪正例FP	真反例TN

图 11-29 准确率示意图

准确率的计算公式为：

$$\frac{TP + TN}{TP + FP + FN + TN}$$

## 3. 召回率

召回率，即真实为正例的样本中预测结果为正例的比例（对正样本的区分能力）。召回率有助于从正例类别的角度评估模型性能，它表明模型能够正确预测出的实际正例样本数占正例样本总数的百分比。比如真实值有 20 个，但是预测出真实值有 16 个，那么召回率为 0.8，如下图 11-30 所示。

批注 [雷4]: 与上文准确率与精确率是否雷同

批注 [HZ5R4]: 不一样，两个概念，一个精准率，一个准确率

批注 [雷6]: 语意不明，需审核修改

		预测结果	
		正例	假例
真实结果	正例	真正例TP	伪反例FN
	假例	伪正例FP	真反例TN

图 11-30 召回率示意图

召回率的计算公式为：

$$\frac{TP}{TP + FN}$$

#### 4. F1-score

F1-score 能够反映模型的稳健型，计算公式为：

$$F1 = \frac{2TP}{2TP + FN + FP} = \frac{2 \cdot \text{精准率} \cdot \text{召回率}}{\text{精准率} + \text{召回率}}$$

#### 5. ROC/AUC

ROC 曲线用于确定模型的阈值，它是召回率和精准率之间的一条曲线，如图 11-31 所示。

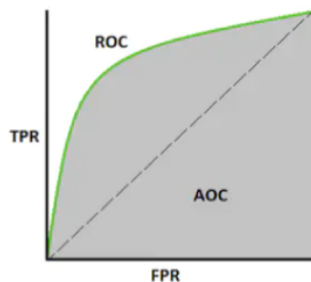


图 11-31 ROC 曲线

其中，横轴为  $FPR = FP/(FP + TN)$ ，为假阳率，即错认为正例的负例占有所有负例的比例；纵轴  $TPR = TP/(TP + FN)$ ，为预测真正例的样本占与所有正例的比例。ROC 是对分类结果描绘出来的曲线，通过阈值的设置可以得到 ROC 曲线，完全正确完美的模型  $FPR = 0, TPR = 1$ ；AUC 为 ROC 曲线下的面积，AUC 的值越接近 1，则表明模型的区分度越好，越接近 0，则越差。

### 11.5.3 创建 SparkSession

在前面小节了解逻辑回归算法原理之后,接下来进入代码实战环节,首先创建本小节的 SparkSession,代码如下图 11-32 所示。

```
In [40]: from pyspark.sql import SparkSession
spark = SparkSession\
    .builder\
    .appName('log_reg')\
    .getOrCreate()

In [41]: spark

Out[41]: SparkSession - hive
SparkContext

Spark UI
Version
v2.4.3
Master
local[*]
AppName
log_reg
```

图 11-32 创建逻辑回归 SparkSession

### 11.5.4 读取数据集

本小节逻辑回归代码使用的数据集为 logit\_regression.csv, 接下来我们进行读取数据集。

步骤 1: 读取数据集

代码如下图 11-33 所示。

```
In [42]: df = spark.read.csv(data_path + 'logit_regression.csv', header=True, inferSchema=True)

In [43]: df.show(3)

+-----+-----+-----+-----+-----+-----+
|Country|Age|Repeat_Visitor|Platform|Web_pages_viewed|Status|
+-----+-----+-----+-----+-----+-----+
| India | 41|          1|  Yahoo|             21|      1|
| Brazil| 28|          1|  Yahoo|              5|      0|
| Brazil| 40|          0| Google|              3|      0|
+-----+-----+-----+-----+-----+-----+

only showing top 3 rows
```

图 11-33 读取数据集

步骤 2: 查看数据结构

同上一小节,我们可以查看原始数据集各个字段的类型,如图 11-34 所示。

```
In [45]: df.printSchema()

root
 |-- Country: string (nullable = true)
 |-- Age: integer (nullable = true)
 |-- Repeat_Visitor: integer (nullable = true)
 |-- Platform: string (nullable = true)
 |-- Web_pages_viewed: integer (nullable = true)
 |-- Status: integer (nullable = true)
```

图 11-34 查看数据集结构

### 11.5.5 数据分析

本小节使用的数据集包含 Country、Age、Platform 等特征，数据集标签字段为 Status，我们将 Country、Platform、Status 三个字段维度进行统计分析，查看各维度下的不同取值的数据量以及均值情况。

步骤 1：查看各维度数据量

我们使用 `groupBy()` 算子查看统计量，这里我们分别对 “Country”、“Platform” 和 “Status” 三个维度进行统计计算，实现代码如下图 11-35 所示。

```
In [47]: import pyspark.sql.functions as F
In [48]: df.groupBy('Country').count().show()
+-----+-----+
| Country|count|
+-----+-----+
| Malaysia| 1218|
| India| 4018|
| Indonesia|12178|
| Brazil| 2586|
+-----+-----+

In [49]: df.groupBy('Platform').count().show()
+-----+-----+
| Platform|count|
+-----+-----+
| Yahoo| 9859|
| Bing| 4360|
| Google| 5781|
+-----+-----+

In [50]: df.groupBy('Status').count().show()
+-----+-----+
| Status|count|
+-----+-----+
| 1|10000|
| 0|10000|
+-----+-----+
```

图 11-35 查看各字段数据量

步骤 2：查看各维度数据均值

同样，这里仍然是对 “Country”、“Platform” 和 “Status” 三个维度进行分别进行均值计算。查看均值的算子为 `mean()`，实现代码如下图 11-36 所示。

```
In [51]: df.groupby('Country').mean().show()
```

Country	avg(Age)	avg(Repeat_Visitor)	avg(Web_pages_viewed)	avg(Status)
Malaysia	27.792282430213465	0.5730706075533661	11.192118226600986	0.6568144499178982
India	27.976654156296664	0.5433051269288203	10.727227476356397	0.6212045793927327
Indonesia	28.43159796354081	0.5207751663363442	9.985711939563148	0.5422893742814913
Brazil	30.274168600154677	0.322892498066512	4.921113689095128	0.038669760247486466

```
In [52]: df.groupby('Platform').mean().show()
```

Platform	avg(Age)	avg(Repeat_Visitor)	avg(Web_pages_viewed)	avg(Status)
Yahoo	28.569226087838523	0.5094837204584644	9.599655137437875	0.5071508266558474
Bing	28.68394495412844	0.4720183486238532	9.114908256880733	0.4559633027522936
Google	28.380038055699707	0.5149628092025601	9.804878048780488	0.5210171250648676

```
In [53]: df.groupby('Status').mean().show()
```

Status	avg(Age)	avg(Repeat_Visitor)	avg(Web_pages_viewed)	avg(Status)
1	26.5435	0.7019	14.5617	1.0
0	30.5356	0.3039	4.5449	0.0

图 11-36 查看各字段数据均值

对数据集进行简单数据统计分析的目的在于观察数据是否均衡，如正负样本量是否出现倾斜，例如正样本数量远远大于负样本或某特征几乎全相同，那么该特征可能选取不合适或没有太大意义。

## 11.5.6 特征工程

步骤 1: StringIndexer 转换 Platform 特征

通过观察数据集字段可知，Platform 特征和 Country 特征为字符串类型，代表平台名称和国家名字，我们需要对字符串特征进行转换，并且对其进行编码，分别使用的方法是 StringIndexer 和 OneHotEncoder。如图 11-37 所示。

```
In [55]: platform_indexer = StringIndexer(
        inputCol="Platform",
        outputCol="Platform_Index"
    ).fit(df)
```

```
In [56]: df = platform_indexer.transform(df)
```

```
In [57]: df.show(5)
```

Country	Age	Repeat_Visitor	Platform	Web_pages_viewed	Status	Platform_Index
India	41	1	Yahoo	21	1	0.0
Brazil	28	1	Yahoo	5	0	0.0
Brazil	40	0	Google	3	0	1.0
Indonesia	31	1	Bing	15	1	2.0
Malaysia	32	0	Google	15	1	1.0

only showing top 5 rows

图 11-37 StringIndexer 特征转换

步骤 2: OneHotEncoder 编码 Platform 特征

对 Platform 进行转换后的数值特征进行编码，实现代码如下图 11-38 所示。

```
In [58]: from pyspark.ml.feature import OneHotEncoder
```

```
In [59]: platform_encoder = OneHotEncoder(
    inputCol="Platform_Index",
    outputCol="Platform_Vector"
)

df = platform_encoder.transform(df)
```

```
In [60]: df.show(5)
```

Country	Age	Repeat_Visitor	Platform	Web_pages_viewed	Status	Platform_Index	Platform_Vector
India	41	1	Yahoo	21	1	0.0	(2, [0], [1.0])
Brazil	28	1	Yahoo	5	0	0.0	(2, [0], [1.0])
Brazil	40	0	Google	3	0	1.0	(2, [1], [1.0])
Indonesia	31	1	Bing	15	1	2.0	(2, [], [])
Malaysia	32	0	Google	15	1	1.0	(2, [1], [1.0])

only showing top 5 rows

图 11-38 OneHotEncoder 特征编码

可以看出，原始的 Platform 特征最终转换成了 Platform\_Vector 特征。

步骤 3: StringIndexer 转换 Country 特征

同样的方法，我们需要对 Country 特征进行转换，如下图 11-39 所示。

```
In [63]: country_indexer = StringIndexer(
    inputCol="Country",
    outputCol="Country_Index"
).fit(df)
```

```
In [64]: df = country_indexer.transform(df)
```

```
In [65]: df.show(5)
```

Country	Age	Repeat_Visitor	Platform	Web_pages_viewed	Status	Platform_Index	Platform_Vector	Country_Index
India	41	1	Yahoo	21	1	0.0	(2, [0], [1.0])	1.0
Brazil	28	1	Yahoo	5	0	0.0	(2, [0], [1.0])	2.0
Brazil	40	0	Google	3	0	1.0	(2, [1], [1.0])	2.0
Indonesia	31	1	Bing	15	1	2.0	(2, [], [])	0.0
Malaysia	32	0	Google	15	1	1.0	(2, [1], [1.0])	3.0

only showing top 5 rows

图 11-39 StringIndexer 特征转换

步骤 4: OneHotEncoder 编码 Country 特征

如下图 11-40 所示。



```
In [66]: country_encoder = OneHotEncoder(
        inputCol="Country_Index",
        outputCol="Country_Vector"
    )

df = country_encoder.transform(df)

In [67]: df.select(['Country', 'Country_Index', 'Country_Vector']).show(5)
```

Country	Country_Index	Country_Vector
India	1.0	(3, [1], [1.0])
Brazil	2.0	(3, [2], [1.0])
Brazil	2.0	(3, [2], [1.0])
Indonesia	0.0	(3, [0], [1.0])
Malaysia	3.0	(3, [1], [])

only showing top 5 rows

图 11-40 OneHotEncoder 特征编码

#### 步骤 5: 合并特征向量

在完成上述特征工程构建之后，我们筛选所需特征，并使用 `VectorAssembler` 方法将特征创建为一个特征向量，如下图 11-41 所示。

```
In [71]: assembler = VectorAssembler(
        inputCols=['Platform_Vector', 'Country_Vector', 'Age', 'Repeat_Visitor', 'Web_pages_viewed'],
        outputCol='features'
    )

df = assembler.transform(df)

In [72]: df.select(
        'Platform_Vector',
        'Country_Vector',
        'Age',
        'Repeat_Visitor',
        'Web_pages_viewed',
        'features',
        'status').show(5, False)
```

Platform_Vector	Country_Vector	Age	Repeat_Visitor	Web_pages_viewed	features	status
(2, [0], [1.0])	(3, [1], [1.0])	41	1	21	[1.0, 0.0, 0.0, 1.0, 0.0, 41.0, 1.0, 21.0]	1
(2, [0], [1.0])	(3, [2], [1.0])	28	1	5	[1.0, 0.0, 0.0, 0.0, 1.0, 28.0, 1.0, 5.0]	0
(2, [1], [1.0])	(3, [2], [1.0])	40	0	3	(8, [1, 4, 5, 7], [1.0, 1.0, 40.0, 3.0])	0
(2, [1], [])	(3, [0], [1.0])	31	1	15	(8, [2, 5, 6, 7], [1.0, 31.0, 1.0, 15.0])	1
(2, [1], [1.0])	(3, [1], [])	32	0	15	(8, [1, 5, 7], [1.0, 32.0, 15.0])	1

only showing top 5 rows

```
In [73]: model_df = df.select(['features', 'Status'])
```

图 11-41 VectorAssembler 合并特征

同样，完成特征工程之后，模型训练所需数据集只需 `features` 特征向量和数据标签 `label`，即 `Status` 字段。

### 11.5.7 构建训练集和测试集

在前面小节中，我们完成了数据集特征工程的构建，接下来我们将对数据集进行切分和模型的训练工作。

#### 步骤 1: 切分数据集

我们按照 7:3 的比例将数据集切分为训练集和测试集，如下图 11-42 所示。

```
In [74]: train_df, test_df = model_df.randomSplit([0.7,0.3])

In [75]: print((train_df.count(), len(train_df.columns)))
(13928, 2)

In [76]: print((test_df.count(), len(test_df.columns)))
(6072, 2)
```

图 11-42 切分数据集

#### 步骤 2: 测试集正负样本量

在这里，我们对测试集的正负样本量进行统计，当然，读者也可以自行对训练集的正负样本量进行查验，如下图 11-43 所示。

```
In [77]: test_df.groupBy('Status').count().show()

+-----+-----+
|Status|count|
+-----+-----+
|      1| 2982|
|      0| 3090|
+-----+-----+
```

图 11-43 测试集正负样本量

从上图可以看出，测试集的正负样本量基本相同，说明切分数据集比较均衡。

### 11.5.8 模型训练与测试

逻辑回归属于分类模型，所以我们需要从 `pyspark.ml.classification` 中引用 `LogisticRegression` 方法，构建逻辑回归模型。

#### 步骤 1: 训练逻辑回归模型

创建 `LogisticRegression` 模型只需要设置一个参数，即 `labelCol`，它标识数据集中的 `label` 标签列。接下来使用 `fit()` 方法即可实现对数据集的训练，最后，使用模型中 `evaluate()` 方法可以实现对数据集的评估。

```
In [79]: from pyspark.ml.classification import LogisticRegression

In [80]: model = LogisticRegression(labelCol='Status')
In [81]: lr_model = model.fit(train_df)
In [82]: train_results = lr_model.evaluate(train_df).predictions
```

设置训练集label列

训练集模型评估

图 11-44 创建逻辑回归模型

从上图代码输出中可以看出以下信息，如线性回归模型的截距以及各维度特征的系数值。

#### 步骤 2: 查看训练集结果

我们通过图 11-45 中的结果可以查看上一步骤中对数据集的评估预测结果。图中 “In[83]” 部分代码和 “In[84]” 部分代码分别筛选出模型预测为正样本和预测为负样本的数据进行展示。可以看出，预测结果筛选出了 3 列数据，分别是 `Status`（代表该行数据的原始标签）、`prediction`（代表模型对该行

批注 [雷7]: 图片上的步骤内容，需在正文中描述出来，全文相同问题同样处理

数据的预测标签)和 probability (该列的数据类型为 List 类型, 包含两个原始, 分别是模型将该行数据预测为负样本的概率和预测为正样本的概率, 当模型预测为正样本的概率大于预测为负样本的概率时, 预测值为正样本标签, 反之亦然)。

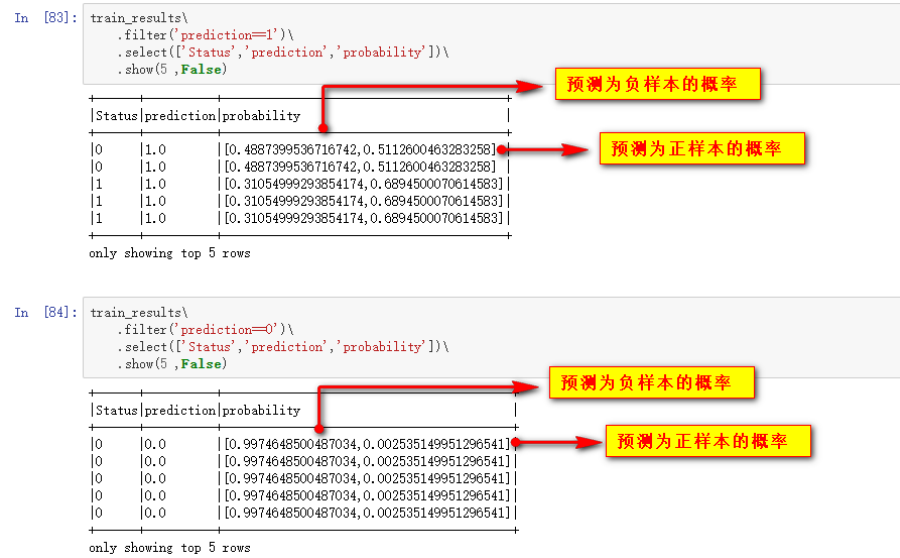


图 11-45 训练结果评估

从上图可以看出模型对每条样本预测的概率。其中 prediction 列为对样本预测的结果, Status 为样本的真实标签, probability 包含两个概率, 分别是预测为负样本和正样本的概率值。若预测结果为负样本的概率大于 0.5, 则预测为负样本, 反之亦然, 当然, 预测为正负样本的概率之和为 1。

步骤 3: 测试集预测结果

我们使用训练好的逻辑回归模型, 在测试集上进行验证。预测结果所包含的数据同上一步中对训练集的评估预测, 这里我们只筛选出 Status 列和 prediction 列进行打印展示。如下图 11-46 所示

```
In [87]: predict = lr_model.evaluate(test_df).predictions

In [88]: predict.printSchema()

root
 |-- features: vector (nullable = true)
 |-- Status: integer (nullable = true)
 |-- rawPrediction: vector (nullable = true)
 |-- probability: vector (nullable = true)
 |-- prediction: double (nullable = false)

In [89]: predict.select(['Status', 'prediction']).show(10, False)
```

Status	prediction
0	0.0
0	0.0
0	0.0
0	0.0
0	0.0
0	0.0
0	0.0
0	0.0
0	0.0
0	0.0
1	0.0

only showing top 10 rows

图 11-46 测试集预测结果

同步步骤 2，我们也可以使用同样的方法对测试集预测结果进行查看。

### 11.5.9 混淆矩阵计算

本小节我们手动计算逻辑回归模型的混淆矩阵和相关模型评估参数。通过手动计算的方式，能够让读者加深对混淆矩阵中各个元素的理解。

#### 步骤 1：混淆矩阵计算

混淆矩阵计算实现代码如下图 11-47 所示。

这里为了帮助读者更好的理解，我们采用手动方式计算混淆矩阵，计算公式如 11.5.2 小节中所介绍。

```
In [90]: from pyspark.ml.evaluation import BinaryClassificationEvaluator

In [91]: #confusion matrix

In [92]: true_positives = predict[(predict.Status == 1) & (predict.prediction == 1)].count()
true_negatives = predict[(predict.Status == 0) & (predict.prediction == 0)].count()
false_positives = predict[(predict.Status == 0) & (predict.prediction == 1)].count()
false_negatives = predict[(predict.Status == 1) & (predict.prediction == 0)].count()
```

图 11-47 混淆矩阵计算

#### 步骤 2：相关评估指标计算

在这里，对于逻辑回归模型我们将分别计算模型的准确率、召回率以及精准率（也叫精度），具体计算代码如下图 11-48 所示，计算公式参考 11.5.2 小节中所介绍。

```

In [93]: #准确率
accuracy=float((true_positives+true_negatives)/(predict.count()))
print("accuracy:", accuracy)

#召回率
recall = float(true_positives)/(true_positives + false_negatives)
print("recall:", recall)

#精度
precision = float(true_positives) / (true_positives + false_positives)
print("precision:", precision)

accuracy: 0.9380764163372859
recall: 0.937625754527163
precision: 0.9363697253851306

```

图 11-48 模型评估指标计算

从以上评估指标可以看出，本节训练的逻辑回归模型较好，相关评价指标也较为理想。

## 11.6 K-Means 聚类

本小节将对 K-Means 聚类方法进行介绍与实战，常见的聚类方法有：K-Means 聚类、层次聚类、密度聚类、谱聚类和高斯混合聚类等。

### 11.6.1 K-Means 聚类介绍

K-Means 算法是一种无监督的聚类算法。K-Means 核心思想是：给定的样本数据集，根据样本点之间的距离大小，把数据集划分成 K 个簇，并让簇内的样本点尽量距离近，而不同簇之间的距离尽可能的远。

K-Means 聚类过程有四个步骤，即聚类簇数 K 值的选择、K 个聚类中心点的初始值选择、距离度量方式、损失函数的选择。

步骤 1：聚类簇数 K 值的选择

聚类簇数 K 值的选择是一个比较难处理的地方，它会对 K-Means 算法的最终结果起到关键作用，在实际中，一般并不知道要把样本数据分为几类。K-Means 算法在处理这个问题时注意依靠人工试探或者超参数探索的形式来确定。

步骤 2：K 个聚类中心点初始值选择

K 个聚类中心点的初始值选择会直接的影响算法需要更新迭代的次数。K-Means 算法是先随机从样本点中选择 K 个点作为聚类中心点的初始值。

步骤 3：距离度量方式

距离的计算有很多方式，如欧式距离、汉明距离等，使用较多的是欧式距离。假设数据样本点的特征维度是 N，那么，两个 N 维向量  $X = (x_{11}, x_{12}, \dots, x_{1n})$  和  $Y = (y_{11}, y_{12}, \dots, y_{1n})$  之间的欧式距离为：

$$d = \sqrt{\sum_{i=1}^N (x_{1i} - y_{1i})^2}$$

#### 步骤 4: 损失函数的选择

判断聚类迭代是否趋于稳定需要根据聚类损失函数的变化情况来判断。聚类算法的损失函数的各个簇中样本向量对应簇均值向量的均方误差。假设，数据样本点为  $X = (x_1, x_2, \dots, x_n)$ ，需要被聚集成  $K$  个簇  $C = (c_1, c_2, \dots, c_n)$ ，则各个簇内样本点的均值向量为：

$$E = \sum_{k=1}^K \sum_{x_i \in C_k} \|x_i - \mu_k\|^2$$

所以，迭代求解 K-Means 算法的目标就是最小化损失函数，即均方误差。

### 11.6.2 创建 SparkSession

本小节将进入 K-Means 聚类实战，首先创建本小节的 SparkSession，代码如下图 11-49 所示。

```
In [95]: from pyspark.sql import SparkSession
spark = SparkSession\
    .builder\
    .appName('k_means')\
    .getOrCreate()

In [96]: spark
Out[96]: SparkSession - hive
SparkContext

Spark UI
Version
v2.4.3
Master
local[*]
AppName
k_means
```

图 11-49 创建 K-Means 聚类 SparkSession

### 11.6.3 读取数据集

本小节 K-Means 聚类算法所使用的数据集为 clustering\_data.csv，接下来我们进行读取数据集。

#### 步骤 1: 读取数据集

该数据集为鸢尾花数据集，是分类算法经常使用的数据集，通过花瓣长度、宽度等特征来区分 3 种类别的花朵，如下图 11-50 所示。

```
In [97]: df = spark.read.csv(data_path + 'clustering_data.csv', header=True, inferSchema=True)
```

```
In [98]: df.show(3)
```

sepal_length	sepal_width	petal_length	petal_width	species
5.1	3.5	1.4	0.2	setosa
4.9	3.0	1.4	0.2	setosa
4.7	3.2	1.3	0.2	setosa

only showing top 3 rows

图 11-50 读取数据集

步骤 2: 查看数据内容

下面, 我们可以查看原始数据集各个字段的类型, 如下图 11-51 所示。

```
In [99]: df.printSchema()
```

```
root
|-- sepal_length: double (nullable = true)
|-- sepal_width: double (nullable = true)
|-- petal_length: double (nullable = true)
|-- petal_width: double (nullable = true)
|-- species: string (nullable = true)
```

图 11-51 查看数据集结构

数据集各个特征均为 double 类型。

接下来, 我们对数据集包含的 3 种类型花朵数量进行统计, 如下图 11-52 所示。

```
In [102]: df.groupBy('species').count().orderBy('count', ascending=False).show(10, False)
```

species	count
versicolor	50
setosa	50
virginica	50

图 11-52 数据集类别统计

可以看出, 数据集包含的 3 种类别花朵的数据各有 50 条。

#### 11.6.4 特征工程

由于数据集的特征字段均为 double 类型, 所以不需要进行额外特征工程, 只需要将 4 个特征合并成一个特征向量即可, 实现代码如下图 11-53 所示。

这里的特征工程同样使用 VectorAssembler 方法将多列特征合成一列特征。

```
In [103]: from pyspark.ml.linalg import Vectors
          from pyspark.ml.feature import VectorAssembler

In [104]: vec_assembler = VectorAssembler(
          inputCols = ['sepal_length', 'sepal_width', 'petal_length', 'petal_width'],
          outputCol='features'
          )

          model_df = vec_assembler.transform(df)

In [105]: model_df.show(5)
```

sepal_length	sepal_width	petal_length	petal_width	species	features
5.1	3.5	1.4	0.2	setosa	[5.1, 3.5, 1.4, 0.2]
4.9	3.0	1.4	0.2	setosa	[4.9, 3.0, 1.4, 0.2]
4.7	3.2	1.3	0.2	setosa	[4.7, 3.2, 1.3, 0.2]
4.6	3.1	1.5	0.2	setosa	[4.6, 3.1, 1.5, 0.2]
5.0	3.6	1.4	0.2	setosa	[5.0, 3.6, 1.4, 0.2]

only showing top 5 rows

图 11-53 合并特征向量

### 11.6.5 模型测试与评估

和其他算法不同，K-Means 算法不需要切分训练集和测试集，同样，K-Means 算法训练的本质即对数据集所有数据进行计算距离/相似度，模型构建具体步骤如下。

步骤 1：创建 K-Means 模型

```
In [106]: from pyspark.ml.clustering import KMeans

In [107]: errors=[]

          for k in range(2,10):
              kmeans = KMeans(featuresCol='features',k=k)
              model = kmeans.fit(model_df)
              intra_distance = model.computeCost(model_df)
              errors.append(intra_distance)

          print('\n'*30)
          print("With K={} ".format(k))
          print("Within Set Sum of Squared Errors = " + str(intra_distance))
```

```
With K=2
Within Set Sum of Squared Errors = 152.36870647734008

With K=3
Within Set Sum of Squared Errors = 78.94506582597637

With K=4
Within Set Sum of Squared Errors = 57.47147508745658

With K=5
Within Set Sum of Squared Errors = 46.53558205128334

With K=6
Within Set Sum of Squared Errors = 45.96395128205201

With K=7
Within Set Sum of Squared Errors = 37.4860212121211

With K=8
Within Set Sum of Squared Errors = 36.35147828282878

With K=9
Within Set Sum of Squared Errors = 34.79115066845024
```



图 11-54 K-Means 模型训练

如上图所示，K-Means 聚类模型的参数即分类数 K，我们将 K 分别取 2~9，通过计算 SSE 的方式来探索模型，可以想象，随着 K 的增大，SSE 会逐渐减小，但是可能会产生过拟合，不难理解，当 K 值接近样本数时，相当于每个样本自己成为一个类别。

#### 步骤 2: K=3 聚类

由于我们知道数据集总共包含 3 类，我们将 K 设置为 3，并进一步查看此时聚类的效果，如下图 11-55 所示。

```
In [110]: kmeans = KMeans(featuresCol='features', k=3)
In [111]: model = kmeans.fit(model_df)
In [112]: model.transform(model_df)\
           .groupBy('prediction')\
           .count()\
           .show()
```

prediction	count
1	50
2	39
0	61

图 11-55 K=3 聚类模型

#### 步骤 3: 查看预测结果

```
In [113]: predicts = model.transform(model_df)
In [114]: predicts.show(5)
```

sepal_length	sepal_width	petal_length	petal_width	species	features	prediction
5.1	3.5	1.4	0.2	setosa	[5.1, 3.5, 1.4, 0.2]	1
4.9	3.0	1.4	0.2	setosa	[4.9, 3.0, 1.4, 0.2]	1
4.7	3.2	1.3	0.2	setosa	[4.7, 3.2, 1.3, 0.2]	1
4.6	3.1	1.5	0.2	setosa	[4.6, 3.1, 1.5, 0.2]	1
5.0	3.6	1.4	0.2	setosa	[5.0, 3.6, 1.4, 0.2]	1

only showing top 5 rows

图 11-56 查看聚类结果

上图中 prediction 列即为模型对每行样本的预测聚类的结果。

#### 步骤 4: 展示聚类结果

上述结果虽然可以看出聚类的结果，但是不够直观，我们可以将预测结果转换为 Pandas DataFrame，并通过 Python 图形绘制的方法更为直观的观察结果，此外，还可以增进读者对 Pyspark DataFrame 与 Pandas DataFrame 交互的理解，关于如何使用 Python matplotlib 绘制图形的方法，本书不再展开赘述，具体实现如下图 11-57 所示。

```
In [116]: pandas_df = predicts.toPandas()
pandas_df.sample(5)
```

转换 Pandas DataFrame

```
Out[116]:
```

	sepal_length	sepal_width	petal_length	petal_width	species	features	prediction
61	5.9	3.0	4.2	1.5	versicolor	[5.9, 3.0, 4.2, 1.5]	0
85	6.0	3.4	4.5	1.6	versicolor	[6.0, 3.4, 4.5, 1.6]	0
137	6.4	3.1	5.5	1.8	virginica	[6.4, 3.1, 5.5, 1.8]	2
33	5.5	4.2	1.4	0.2	setosa	[5.5, 4.2, 1.4, 0.2]	1
129	7.2	3.0	5.8	1.6	virginica	[7.2, 3.0, 5.8, 1.6]	2

```
In [117]: import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
```

```
In [118]: cluster_vis = plt.figure(figsize=(10,8)).gca(projection='3d')
cluster_vis.scatter(
    pandas_df.sepal_length,
    pandas_df.sepal_width,
    pandas_df.petal_length,
    c=pandas_df.prediction,
    depthshade=False)
plt.show()
```

绘制3D图形

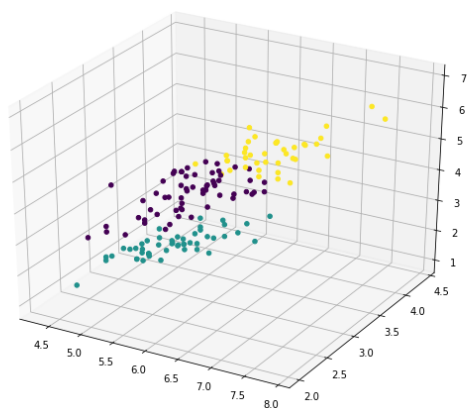


图 11-57 绘制聚类结果图形

通过 3D 图形绘制可以直观的看出 3 种类别的聚类情况，图中每种颜色代表一个聚类，每个颜色类别的点越紧密，且能够明显区分其他颜色点的范围，则表明聚类的效果较好。

## 11.7 随机森林

### 11.7.1 决策树介绍

决策树是一种有监督的机器学习的方法。决策树的生成算法有 ID3、C4.5 和 C5.0 等。决策树是一种树形结构，它是由节点和有向边组成的层次结构。树中包含 3 种节点：根节点、内部节点、叶子节点。决策树只有一个根节点，是全体训练数据的集合。树中每个内部节点都是一个分裂问题：指定了对实例的某个属性的测试，它将到达该节点的样本按照某个特定的属性进行分割，并且该节点的每

一个后继分支对应于该属性的一个可能值。每个叶子节点是带有分类标签的数据集合即为实例所属的分类。

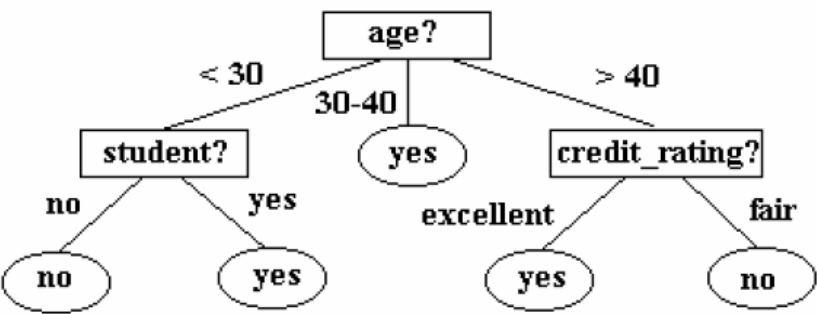


图 11-58 典型的决策树

上图表示概念 `buys_compute`，即它目的是预测顾客是否可能购买计算机。内部节点用矩形表示，叶子节点用椭圆表示。

为了对未知的样本分类，样本的属性值在决策树上测试。每个内部节点表示一个属性上的测试，每个叶子节点代表一个分类（例如，`buys_computer = yes`, `buys_computer = no`）。

决策树很重要的一点是决策属性的选择。建树算法中属性的选择非常重要。属性选择方法很多种，例如信息增益（Information Gain）、信息增益比（Information Gain Ratio）、Gini 指标（Gini Index）等方法。

限于篇幅，本书不对决策树的相关数学推导展开赘述。

11.7.2 随机森林介绍

在上一小节中，我们了解了决策树的相关概念，本小节继续对随机森林进行介绍。随机森林，一种集成算法（Ensemble Learning），它属于 Bagging 类型，随机森林模型即可以用于解决回归问题，也可以用于解决分类问题，通过组合多个弱分类器，最终结果通过投票或取均值，使得整体模型的结果具有较高的精确度和泛化性能。其可以取得不错成绩，主要归功于“随机”和“森林”，一个使它具有抗过拟合能力，一个使它更加精准，如下图 11-59 所示。

随机森林，从名字上就可以看出，它有很多很多树组成，从而构成了森林，那么这里的“树”，或者说前面提到的弱分类器，即决策树。

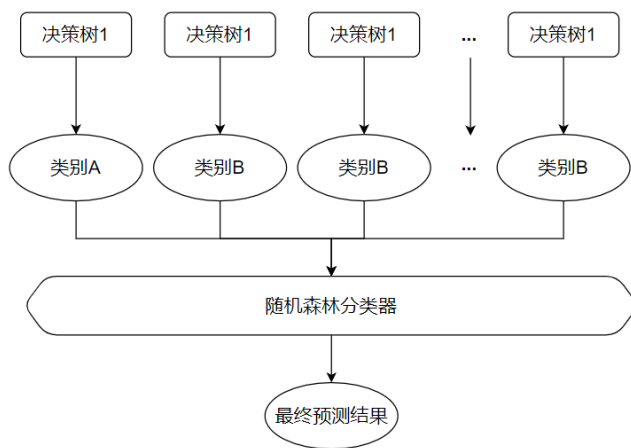


图 11-59 随机森林示意图

随机森林与决策树相比有如下优势：

1. 特征重要性，随机森林可以提供被用于训练的每一个特征的重要性，使得可以选取相关特征以及丢弃较弱的特征。
2. 预测准确率，由于随机森林会对各个子树模型进行投票，所以相比较于决策树，随机森林的预测能力更高一些。
3. 不易过拟合，由于各个子分类器的结果会被通过随机森林分类器进行投票表决，所以会降低某个树产生过拟合而导致模型整体过拟合的概率。
4. 模型性能，随机森林模型训练速度快，容易并行化计算。

当然，随机森林模型也不是完美的模型，同样存在缺点：

1. 在某些噪音比较大的样本集上，随机森林模型容易陷入过拟合。
2. 取值划分比较多的特征容易对 RF 的决策产生更大的影响，从而影响拟合的模型的效果。

### 11.7.3 创建 SparkSession

本小节将进入随机森林聚类实战，首先创建本小节的 SparkSession，代码如下图 11-60 所示。

```
In [120]: from pyspark.sql import SparkSession
spark = SparkSession\
    .builder\
    .appName('Random_Forests')\
    .getOrCreate()
```

```
In [121]: spark
```

```
Out[121]: SparkSession - hive
SparkContext

Spark UI
Version
v2.4.3
Master
local[*]
AppName
Random_Forests
```

图 11-60 创建随机森林 SparkSession

### 11.7.4 读取数据集

在前面小节中，随机森林实战使用的数据集文件为 `random_forests.csv`，包含 `rate_marriage`、`age`、`yrs_married`、`children`、`religious` 特征。

步骤 1：读取数据集

代码如图 11-61 所示。

```
In [122]: df = spark.read.csv(data_path + 'random_forests.csv', header=True, inferSchema=True)
In [123]: df.show(3)
```

rate_marriage	age	yrs_married	children	religious	affairs
5	32.0	6.0	1.0	3	0
4	22.0	2.5	0.0	2	0
3	32.0	9.0	3.0	3	1

only showing top 3 rows

图 11-61 读取数据集

步骤 2：查看数据内容

接下来，我们可以查看原始数据集各个字段的类型，如图 11-62 所示。

```
In [124]: df.printSchema()
```

```
root
|-- rate_marriage: integer (nullable = true)
|-- age: double (nullable = true)
|-- yrs_married: double (nullable = true)
|-- children: double (nullable = true)
|-- religious: integer (nullable = true)
|-- affairs: integer (nullable = true)
```

图 11-62 查看数据集结构

### 11.7.5 数据分析

本小节我们通过数据统计的方法更进一步了解数据集内容。

步骤 1：查看数据集统计指标

我们使用 `describe()` 方法检查数据集的统计指标，如图 11-63 所示。

```
In [126]: df.describe().select('summary', 'rate_marriage', 'age', 'yrs_married', 'children', 'religious').show(5)
```

	summary	rate_marriage	age	yrs_married	children	religious
count	6366	6366	6366	6366	6366	6366
mean	4.109644989004084	29.082862079798932	9.00942507068803	1.3968740182218033	2.4261702796104303	
stddev	0.9614295945655025	6.847881883668817	7.280119972766412	1.433470828560344	0.8783688402641785	
min	1	17.5	0.5	0.0	1	
max	5	42.0	23.0	5.5	4	

图 11-63 查看数据集统计指标

步骤 2: 查看正负样本量

我们使用 `groupBy()` 方法对 “affairs” 列进行聚合统计，如图 11-64 所示。

```
In [127]: df.groupBy('affairs').count().show()
```

affairs	count
1	2053
0	4313

图 11-64 查看正负样本量

## 11.7.6 特征工程

同上一小节，本节随机森林使用的数据集字段类型为整型或浮点数类型，没有字符串类型特征，所以只需将特征合并成一个特征向量即可，实现代码如下图 11-65 所示。

步骤 1: `VectorAssembler` 合并特征向量

```
In [131]: from pyspark.ml.feature import VectorAssembler
```

```
In [132]: vec_assembler = VectorAssembler(
    inputCols=['rate_marriage', 'age', 'yrs_married', 'children', 'religious'],
    outputCol="features"
```

```
)

In [133]: df = vec_assembler.transform(df)
```

```
In [134]: df.show(5)
```

rate_marriage	age	yrs_married	children	religious	affairs	features
5	32.0	6.0	1.0	3	0	[5.0, 32.0, 6.0, 1.0, 3.0, 0.0]
4	22.0	2.5	0.0	2	0	[4.0, 22.0, 2.5, 0.0, 2.0, 0.0]
3	32.0	9.0	3.0	3	1	[3.0, 32.0, 9.0, 3.0, 3.0, 1.0]
3	27.0	13.0	3.0	1	1	[3.0, 27.0, 13.0, 3.0, 1.0, 1.0]
4	22.0	2.5	0.0	1	1	[4.0, 22.0, 2.5, 0.0, 2.0, 1.0]

only showing top 5 rows

图 11-65 合并特征向量

步骤 2: 构建数据集

```
In [135]: model_df = df.select(['features', 'affairs'])

In [136]: model_df.printSchema()

root
 |-- features: vector (nullable = true)
 |-- affairs: integer (nullable = true)
```

图 11-66 构建数据集

我们仅需将新构建的 `features` 向量特征和标签字段 `affairs` 筛选出来用于模型训练即可，如上图 11-66 所示。

### 11.7.7 构建训练集和测试集

#### 步骤 1：切分数据集

我们将对数据集进行切分和模型的训练工作，按照 7:3 的比例将数据集切分为训练集和测试集。具体实现代码如图 11-67 所示。

```
In [137]: train_df, test_df = model_df.randomSplit([0.7, 0.3])

In [138]: train_df.count(), test_df.count()

Out[138]: (4417, 1949)
```

图 11-67 切分数据集

#### 步骤 2：查看正负样本量

接下来，我们对训练集和测试集正负样本的数量分别进行统计查看，如图 11-68 所示。

```
In [139]: train_df.groupBy('affairs').count().show()

+-----+-----+
|affairs|count|
+-----+-----+
|      1| 1446|
|      0| 2971|
+-----+-----+

In [140]: test_df.groupBy('affairs').count().show()

+-----+-----+
|affairs|count|
+-----+-----+
|      1|  607|
|      0| 1342|
+-----+-----+
```

图 11-68 查看正负样本量

### 11.7.8 模型训练测试

#### 步骤 1：创建随机森林模型

随机森林模型位于 `pyspark.ml` 分类模型中，我们创建随机森林模型并进行训练和预测。该模型只需要设置两个参数，分别是 `labelCol`，即数据集的 `label` 标签列；另一个参数是 `numTrees`，即随机森林模型中树的个数，注意，该参数需要根据数据集进行调整，设置太大可能会导致模型过拟合，反之，如果设置的太小，模型可能学不到有用知识，导致模型欠拟合，具体实现代码如图 11-69 所示。

```
In [141]: from pyspark.ml.classification import RandomForestClassifier


In [142]: rf_classifier = RandomForestClassifier(
            labelCol='affairs',
            numTrees=50
        )
rf_model = rf_classifier.fit(train_df)
```

图 11-69 创建模型和训练

## 步骤 2：模型预测

使用训练好的模型对测试集进行预测，代码如下图 11-70 所示。

```
In [143]: predicts = rf_model.transform(test_df)
```

 **测试集预测**

```
In [144]: predicts.printSchema()
```

**预测结果数据结构**

```
root
 |-- features: vector (nullable = true)
 |-- affairs: integer (nullable = true)
 |-- rawPrediction: vector (nullable = true)
 |-- probability: vector (nullable = true)
 |-- prediction: double (nullable = false)
```

```
In [145]: predicts.select('affairs','features','rawPrediction','probability','prediction').show(5)
```

affairs	features	rawPrediction	probability	prediction
1	[1.0, 22.0, 2.5, 1.0...]	[19.6265267888139...]	[0.39253053577627...]	1.0
1	[1.0, 22.0, 2.5, 1.0...]	[19.6265267888139...]	[0.39253053577627...]	1.0
0	[1.0, 22.0, 2.5, 1.0...]	[19.2068509810410...]	[0.38413701962082...]	1.0
1	[1.0, 27.0, 6.0, 1.0...]	[17.0007472438811...]	[0.34001494487762...]	1.0
0	[1.0, 27.0, 6.0, 1.0...]	[16.8786238710087...]	[0.33757247742017...]	1.0

only showing top 5 rows

图 11-70 模型预测结果

其中，第一列（affairs）代表数据的真实标签；第二列（features）代表数据的输入特征向量；第三列（rawPrediction）代表两个可能输出的置信度指标；第四列（probability）代表每个类别标签的条件概率；最后一列（prediction）代表模型预测的结果。

## 步骤 3：特征重要性

如前面小节中介绍的随机森林算法，随机森林模型可以提供每个特征的重要性，有助于反映对预测能力贡献最大的关键特征。

```
In [147]: rf_model.featureImportances
```

```
Out[147]: SparseVector(5, [0: 0.6524, 1: 0.0236, 2: 0.2025, 3: 0.069, 4: 0.0524])
```

```
In [148]: model_df.schema["features"].metadata["ml_attr"]["attrs"]
```

```
Out[148]: {'numeric': [{'idx': 0, 'name': 'rate_marriage'},
                    {'idx': 1, 'name': 'age'},
                    {'idx': 2, 'name': 'yrs_married'},
                    {'idx': 3, 'name': 'children'},
                    {'idx': 4, 'name': 'religious'}]}
```

图 11-71 特征重要性

模型使用了 5 个特征，可以使用特征重要性函数来查看特征的重要程度。如果需要了解某个特征的元数据信息，可以通过 metadata 来查看，如上图 11-71 所示。

从本小节实例来看，输入特征中最重要的是 rate\_marriage 特征。



### 11.7.9 评估指标分析

接下来，我们仍需要通过指标来分析随机森林模型的效果。

#### 步骤 1：准确率评估

这里我们使用多分类评估器来评估模型的准确率，`MutiClassificationEvaluator` 中包含两个参数，分别是 `labelCol`，即标签列列名；另一个是 `metricName`，即需要采用那种评估方法，对于准确率评估这里我们使用 “accuracy”，具体实现代码如图 11-72 所示。

```
In [149]: from pyspark.ml.evaluation import BinaryClassificationEvaluator
          from pyspark.ml.evaluation import MulticlassClassificationEvaluator

In [150]: evaluator = MulticlassClassificationEvaluator(
          labelCol='affairs',
          metricName='accuracy')

In [151]: accuracy = evaluator.evaluate(predicts)

In [152]: print('The accuracy of RF on test data is ', round(accuracy, 3))

The accuracy of RF on test data is  0.725
```

引入模型评估相关函数

批注 [雷8]: 图片内容需描述在正文中

图 11-72 模型准确率评估

#### 步骤 2：精准率评估

进行准确率评估只需要将步骤 1 中 `metricName` 参数的值设置为 “weightedPrecision” 即可。

```
In [153]: evaluator = MulticlassClassificationEvaluator(
          labelCol='affairs',
          metricName='weightedPrecision')

In [154]: precision = evaluator.evaluate(predicts)

In [155]: print('The precision rate on test data is ', round(precision, 3))

The precision rate on test data is  0.707
```

批注 [雷9]: 图片内容需描述在正文中

图 11-73 模型精准率评估

#### 步骤 3：AUC 评估

AUC 评估我们使用 `BinaryClassificationEvaluator` 评估器来进行计算，`BinaryClassificationEvaluator` 只需要设置 `labelCol` 参数即可，具体代码如图 11-74 所示。

```
In [156]: evaluator = BinaryClassificationEvaluator(
          labelCol='affairs')

In [157]: auc = evaluator.evaluate(predicts)

In [158]: print('The auc rate on test data is ', round(auc, 3))

The auc rate on test data is  0.742
```

批注 [雷10]: 图片内容需描述在正文中

图 11-74 模型 AUC 评估

### 11.7.10 模型保存与导入

最后，在实际应用中，当我们完成模型训练与评估工作之后，我们通常希望将模型进行保存，当需要预测时直接调用，避免重新再次训练模型。那么，本小节将介绍模型的保存和导入方法。

### 步骤 1: 模型保存

对模型使用 `save()` 函数可以实现模型的保存, 该方法的参数为要存储模型的路径, 存储完成后, 我们可以使用 `%ll` 魔法命令来进行查看, 如下图 11-75 所示。

```
In [160]: rf_model.save(data_path + 'RF_model')

In [161]: %ll ./data/ai/RF_model

总用量 8
drwxr-xr-x. 2 root 4096 2月  5 20:03 data/
drwxr-xr-x. 2 root  84 2月  5 20:03 metadata/
drwxr-xr-x. 2 root 4096 2月  5 20:03 treesMetadata/
```

图 11-75 模型保存

本例展示模型文件存储在本地文件目录, 可以看到, 随机森林模型保存之后包含 `data`、`metadata` 以及 `treesMetadata` 三个文件夹。

### 步骤 2: 导入模型

对于随机森林模型, 导入模型需要引入 `RandomForestClassificationModel` 方法, 具体代码如下图 11-76 所示。

```
In [162]: from pyspark.ml.classification import RandomForestClassificationModel
In [163]: rf = RandomForestClassificationModel.load(data_path + 'RF_model')
In [164]: rf

Out[164]: RandomForestClassificationModel (uid=RandomForestClassifier_5ef2780a4b05) with 50 trees
```

图 11-76 导入保存

### 步骤 3: 模型预测

最后, 我们使用导入的模型进行数据预测, 我们使用导入的模型对测试集进行预测, 如下图 11-77 所示。

```
In [165]: model_predicts = rf.transform(test_df)

In [166]: model_predicts.select('affairs', 'features', 'rawPrediction', 'probability', 'prediction').show(5)
```

affairs	features	rawPrediction	probability	prediction
1	[1.0, 22.0, 2.5, 1.0...]	[19.6265267888139...]	[0.39253053577627...]	1.0
1	[1.0, 22.0, 2.5, 1.0...]	[19.6265267888139...]	[0.39253053577627...]	1.0
0	[1.0, 22.0, 2.5, 1.0...]	[19.2068509610410...]	[0.38413701962082...]	1.0
1	[1.0, 27.0, 6.0, 1.0...]	[17.0007472438811...]	[0.34001494487762...]	1.0
0	[1.0, 27.0, 6.0, 1.0...]	[16.8786238710087...]	[0.33757247742017...]	1.0

only showing top 5 rows

图 11-77 模型预测

## ★回顾思考★

### 01 Spark Mlib 支持的数据类型有哪些

答：

- (1) Local vector
- (2) Labeled point
- (3) Local matrix
- (4) Distributed matrix

### 02 Spark ML 和 Spark Mlib 的区别？

答：

- (1) ML 和 Mlib 都是 Spark 中的机器学习库，目前常用的机器学习功能 2 个库都能满足需求。
- (2) ML 是升级版的 MLlib，最新的 Spark 版本优先支持 ML。
- (3) 二者面向的数据集不一样，Mlib 操作的是 RDD，而 ML 主要操作的是 DataFrame。
- (4) 相比于 Mlib 在 RDD 提供的基础操作，ML 在 DataFrame 上的抽象级别更高，数据和操作耦合度更低。
- (5) ML 支持 Pipelines，可以把很多操作（算法/特征提取/特征转换）以管道的形式串起来。
- (6) ML 中的随机森林支持更多的功能：包括重要度、预测概率输出等，而 MLlib 不支持。

## ★练习★

### 一、选择题

- 1. 不属于逻辑回归模型说明的是（ ）
  - A. 逻辑回归模型可用混淆矩阵进行评估
  - B. 逻辑回归模型是分类模型
  - C. 逻辑回归模型是回归模型
  - D. 逻辑回归模型输出取值为 0~1
- 2. 向量标签格式为（ ）
  - A. label [key1:value1 key2:value2 key3:value3 ...]
  - B. label {key1:value1 key2:value2 key3:value3 ...}
  - C. label key1:value1 key2:value2 key3:value3 ...
  - D. label:label key1:value1 key2:value2 key3:value3 ...
- 3. 逻辑回归模型是线性回归模型使用什么函数进行的非线性变换（ ）
  - A. 指数函数
  - B. sigmod 函数
  - C. 狄利克雷函数
  - D. 对数函数
- 3. 关于 K-Means 聚类算法的说法不正确的是（ ）
  - A. K-Means 算法需要切分训练集和测试集训练
  - B. K-Means 算法是无监督算法
  - C. K-Means 算法是计算每个样本之间的距离进行聚类

D. K-Means 的参数 K 是聚类的个数，需要调试

3. 关于随机森林算法的说法正确的是（ ）

- A. 在某些噪音比较大的样本集上随机森林算法效果不受影响
- B. 随机森林算法可以容易并行化计算
- C. 随机森林算法不能用于回归任务
- D. 随机森林算法不存在过拟合问题

## 二、判断题

- 1. Spark Mlib 操作的是 DataFrame, Spark ML 操作的是 RDD。 ( )
- 2. 逻辑回归是回归模型，不可用于分类算法。 ( )
- 3. 随机森林模型即可以用于分类任务，也可以用于回归任务。 ( )

## 三、实战练习

**任务 1:** 手动实现 11.4~11.7 小节的代码练习。

请参考 11.4~11.7 小节中的代码。

**任务 2:** 保存并导入 11.5 小节训练的逻辑回归模型并用于预测。

请参考本书配套的第 11 章节的 Notebook 代码文件。

批注 [雷11]: 需完成答案

## 本章小结

本章介绍了关于 Spark 机器学习的相关内容，机器学习是 Spark 的核心部分，也是大数据分析处理的一大利器。本章首先对 Mlib 的多种数据类型以及数理统计相关的基本统计量进行了介绍，通过对基本数据类型的知识概念进行了详细讲解，方便读者在后续学习或者工作中灵活的选取合理的数据类型，提高计算效率。然后，本章对常用的 Spark ML 机器学习库中的算法模型进行了实战讲解。掌握这些常用的机器学习方法在后续大数据分析处理中将大有裨益。