第 12 章 Spark ML Pipeline 让机器学习更顺畅

★本章导读★

本章将通过学习 Spark ML Pipeline，将数据处理、机器学习串联起来。Spark ML Pipeline 的引入，是受到 scikit-learn 的启发，虽然 MLlib 已经足够简单实用，但如果目标数据集结构复杂，需要多次处理，或是在学习过程中，要使用多个转化器 (Transformer) 和预测器 (Estimator)，这种情况下使用 MLlib 将会让程序结构极其复杂。所以，一个可用于构建复杂机器学习工作流应用的新库出现了，Spark ML Pipeline 提供了一套基于 DataFrames 构建的统一的高级 API ，可帮助用户创建和调整实用的机器学习管道。

★知识要点★

通过本章内容的学习，读者将掌握以下知识：

- Spark ML Pipeline 的作用与特点
- Spark ML Pipeline 的使用方法
- 使用 Spark ML Pipeline 完成回归算法实战

12.1 Pipeline 主要概念

12.1.1 Pipeline 简介

Spark 机器学习建议使用 ML Pipeline，一般称为管道，用于将多种算法更容易地组合成单个管道流水线或工作流程。一个 Pipeline 在结构上会包含一个或多个 Stage，每一个 Stage 都会完成一个任务，如数据处理、数据转化、模型训练、参数设置或数据预测等，其中两个主要的 Stage 为 Transformer 和 Estimator。Transformer 主要是用来操作一个 DataFrame 数据并生成另外一个 DataFrame 数据，比如决策树模型、特征提取等。Transformer。Estimator 则主要是用来做模型拟合，用来生成一个 Transformer。这些 Stage 有序组成一个 Pipeline。ML Pipeline 的主要概念有: DataFrame、 Transformer、Estimator、Parameter 等。

12.1.2 Pipeline 组件

上一小节中对 ML Pipeline 进行了介绍，并介绍了 ML Pipeline 的主要组件。在本小节中将对 ML Pipeline 的组件进行详细介绍。

1. DataFrame

在前面章节中，我们介绍了 Spark DataFrame 数据集，并对 DataFrame 的使用展开了详细介绍。同样，Spark ML API 使用 Spark SQL 中的 DataFrame 作为机器学习数据集，可以容纳很多种类型的数据，

例如：文本、向量、图像和结构化数据等等。Spark DataFrame 以 RDD(Resilient Distributed Datasets)为基础，但是带有 Schema 结构信息，这样开发者可以更方便的使用数据集进行相应处理操作。

2. Transformer

Transformer 一般翻译成转换器，是一个 Pipeline Stage，转换器包含特征变化和学习模型。从技术上来说，转化器通过方法 transform()，在原始数据上增加一列或者多列来将一个 DataFrame 转为另一个 DataFrame。如:

（1） 一个特征转换器输入一个 DataFrame，读取一个文本列，将其映射为新的特征向量列。输出一个新的带有特征向量列的 DataFrame。

（2） 一个学习模型转换器输入一个 DataFrame，读取包括特征向量的列，预测每一个特征向量的标签，输出一个新的带有预测标签列的 DataFrame。

3. Estimator

Estimator 可以被翻译成评估器或适配器，在 Pipeline 里通常是被用来操作 DataFrame 数据并生产一个 Transformer，一个分类算法就是一个 Estimator。因为它可以通过训练特征数据而得到一个分类模型。估计器用来拟合或者训练数据的学习算法或者任何算法。Estimator 通过调用 fit()方法，接受一个 DataFrame 产生一个模型，这个模型就是一个 Transformer。比如，逻辑回归就是一个 Estimator ，通过调用 fit()来产生一个逻辑回归模型。

4. Pipeline

将 Pipeline 多个 Transformers 和 Estimators 链接在一起以指定 ML 工作流。在机器学习中，通常运行一系列算法来处理和学习数据。例如，一个简单的文本文档处理工作流程可能包括以下几个阶段：

（1） 将每个文档的文本拆分为单词；

（2） 将每个文档的单词转换为数字特征向量；

（3） 使用特征向量和标签学习预测模型。

Spark ML 将这样的工作流表示为一个 Pipeline，它由以特定顺序运行的一系列 PipelineStage（Transformer 和 Estimator）按照特定的顺序运行。

12.1.3 Pipeline 工作流程

要构建一个 Pipeline，首先需要定义 Pipeline 中的各个 stage，如指标提取和转换模型训练等。有了这些处理特定问题的 Transformer 和 Estimator，我们就可以按照具体的处理逻辑来有序地组织 Stage 并创建一个 Pipeline。Pipeline 由一系列有顺序的 stage 指定，输入的 DataFrame 通过每个 stage 进行处理、转换。在 Transformer 阶段，transform()方法被调用于 DataFrame 上。在 Estimator 阶段，fit()方法被调用来产生一个 Transformer，然后该转换器的 transform()方法被调用在 DataFrame 上。下图 12-1 简单说明了文档处理工作流的运行过程。

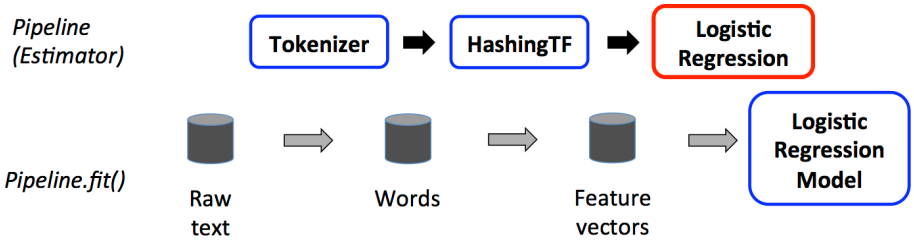

图 12-1 逻辑回归 Pipeline 工程流程

在上图中，第一行代表 Pipeline 处理的三个阶段。第一、二个阶段是 Transformer，第三个逻辑回归是 Estimator。第二行代表 Pipeline 中的数据流，圆柱块即指 DataFrame 数据。通过调用 pipeline.fit()方

法处理、转换 DataFrame。在该示例中，DataFrame 包含原始的文档数据内容，Tokenizer.transform()方法将原始文档分为词语，将新的词语列添加到 DataFrame 中。HashingTF.transform()方法将词语列转换为特征向量，同样添加新的向量列到 DataFrame 中。接下来，因为逻辑回归是 Estimator，Pipeline 会先调用逻辑回归的 fit()方法来产生逻辑回归模型。如果 Pipeline 还有其他更多 stage，则会在 DataFrame 传入下一个 stage 之前，先调用逻辑回归模型的 transform()方法。整个 Pipeline 是一个 Estimator。所以当 Pipeline 的 fit()方法运行后，会产生一个 PipelineModel，PipelineModel 是一个 Transformer，在测试环节之后可以直接调用 transform()方法生成最终结果。

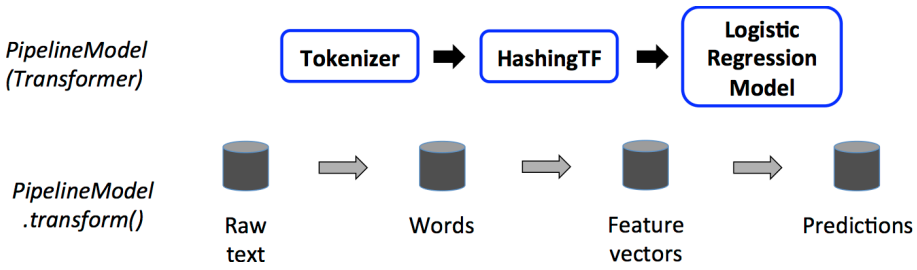

图 12-2 逻辑回归 PipelineModel 工作流程

在上图 12-2 中，PipelineModel 和原始 Pipeline 有同样数目的 stage，但是，原始 Pipeline 中的 Estimator 此时变为了 Transformer。当 PipelineModel 的 transform()方法被调用于测试数据集时，数据依次经过 Pipeline 的各个 stage。每个 stage 的 transform()方法更新数据集，并将之传到下个 stage。Pipeline 和 PipelineModel 能够确保训练数据和测试数据经过同样的特征处理流程。以上两图如果合并为一图，可用下图 12-3 来表达。

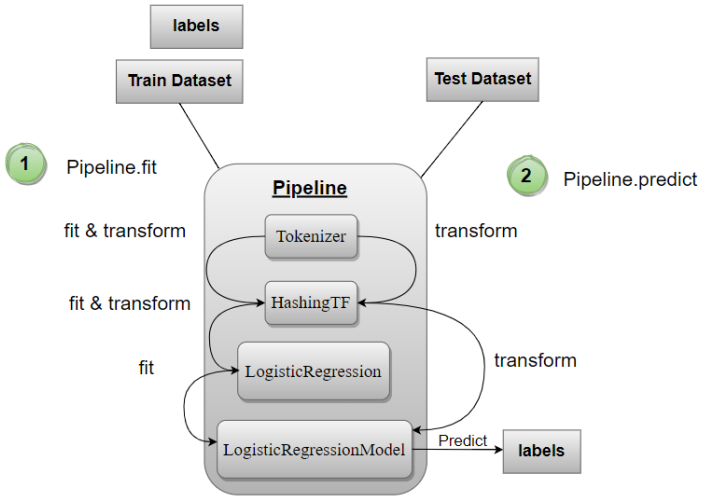

图 12-3 Pipeline 工作流程图

12.2 ML Pipeline 常用方法

12.2.1 StringIndexer

StringIndexer 用于将文字/字符串的离散类别特征转换成数字。

StringIndexer 是一个 Estimator，所以使用上必须包含两个步骤，即：

1. 使用 fit()方法传入参数 DataFrame 数据集，产生一个 Transformer。
2. 使用上一步产生的 Transformer 的方法将 DataFrame 转换成另外一个 DataFrame 数据集。

StringIndexer 需要通过如下代码导入模块。

```
from pyspark.ml.feature import StringIndexer
```

StringIndexer 参数说明：

- inputCol：要进行转换的字段名（列名）
- outputCol：转换后输出的字段名（列名）

StringIndexer 使用方法如图 12-1 中“In[4]”部分代码所示。首先创建名为 df 的 DataFrame，其中包含 id 和 category 两列数据，通过使用 StringIndexer 库实现将 category 类别列转换为数字索引列。

批注 [雷1]: 需核对是否准确

```
In [2]: df = spark.createDataFrame(
            [(0, "a"), (1, "b"), (2, "c"), (3, "a"), (4, "a"), (5, "c")],
            ["id", "category"])

        df.show()
```

1. 创建DataFrame数据集

```
+---+--------+
| id|category|
+---+--------+
|  0|       a|
|  1|       b|
|  2|       c|
|  3|       a|
|  4|       a|
|  5|       c|
+---+--------+
```

类别特征列

2. 定义一个“category”的StringIndexer

```
In [4]: indexer = StringIndexer(inputCol="category", outputCol="categoryIndex")
        indexed = indexer.fit(df)\
                .transform(df)
        indexed.show()
```

3.使用fit()方法生成一个Transformer

4. 使用transform方法进行转换原DataFrame

```
+---+--------+-------------+
| id|category|categoryIndex|
+---+--------+-------------+
|  0|       a|          0.0|
|  1|       b|          2.0|
|  2|       c|          1.0|
|  3|       a|          0.0|
|  4|       a|          0.0|
|  5|       c|          1.0|
+---+--------+-------------+
```

图 12-5 StringIndexer 创建与使用

12.2.2 OneHotEncoder

OneHotEncoder 用于将一个数值类型的特征字段转换为多个字段的 Vector。

OneHotEncoder 将表示为标签索引的分类特征映射到二进制向量，其中最多有一个单值表示所有特征值集中存在特定特征值。对于字符串类型的输入数据，通常首先使用 StringIndexer 对分类特征进行编码。

OneHotEncoder 需要通过如下代码导入模块。

```
from pyspark.ml.feature import OneHotEncoder
```

OneHotEncoder 参数说明：

- inputCol：要进行转换的字段名（列名），可以传多列
- outputCol：转换后输出的字段名（列名），可以传多列
- dropLast：默认为 True，是否包含最后一个类别

OneHotEncoder 使用方法如图 12-6 代码所示。

```
In [23]: df = spark.createDataFrame([
             (0.0, 1.0),
             (1.0, 0.0),
             (2.0, 1.0),
             (0.0, 2.0),
             (0.0, 1.0),
             (2.0, 0.0)
         ], ["categoryIndex1", "categoryIndex2"])

         encoder = OneHotEncoder(inputCol="categoryIndex1",
                                 outputCol="categoryVec1",
                                 dropLast=False
                                )
         encoded = encoder.transform(df)
         encoded.show()
```

1. 创建DataFrame

2. 创建OneHotEncoder

3. 使用transform()方法进行转换

```
+--------------+--------------+--------------+
|categoryIndex1|categoryIndex2| categoryVec1|
+--------------+--------------+--------------+
|           0.0|           1.0|(3,[0],[1.0])|
|           1.0|           0.0|(3,[1],[1.0])|
|           2.0|           1.0|(3,[2],[1.0])|
|           0.0|           2.0|(3,[0],[1.0])|
|           0.0|           1.0|(3,[0],[1.0])|
|           2.0|           0.0|(3,[2],[1.0])|
+--------------+--------------+--------------+
```

图 12-6 OneHotEncoder 的创建与使用

在代码中，输入字段名为 categoryIndex1，转换后的字段名为 categoryVec1，例如第一项数据为 0.0，转换后为 vector([3, [0], [1.0]])。这种格式是我们在上一章节中提到的 Vector 格式中的 SparesVector 格式，其含义表示：第一个数字 3 代表一共编码为 3 位，第二个[0]代表第 0 位是 1，其余是 0，即 0,0,1。使用这种格式存储数据的优点在于可以节省存储空间。

12.2.3 VectorAssembler

VectorAssembler 可以将多个字段特征整合成一个特征向量 Vector。

VectorAssembler 是将给定的列表组合成单个向量列的转换器。它对于将原始特征和不同特征转换器生成的特征组合成单个特征向量很有用，以便训练 ML 模型，如逻辑回归和决策树。 VectorAssembler 接受以下输入列类型：所有数值类型、布尔类型和向量类型。在每一行中，输入列的值将按指定顺序连接成一个向量。

例如，假设我们有一个包含 id、hour、mobile、userFeatures 和列的 DataFrame clicked，如图 12-7 所示：

```
 id | hour | mobile | userFeatures     | clicked
----|------|--------|------------------|---------
 0  | 18   | 1.0    | [0.0, 10.0, 0.5] | 1.0
```

图 12-7 用户特征示例

userFeatures 是一个包含三个用户特征的向量列。我们想将 hour、mobile 和组合 userFeatures 成一个单独的特征向量 features，并用它来预测 clicked 或不预测。如果我们将 VectorAssembler 的输入列设置为 hour、mobile 和 userFeatures 输出列 features，则转换后我们应该得到以下 DataFrame，如图 12-8 所示：

```
 id | hour | mobile | userFeatures     | clicked | features
----|------|--------|------------------|---------|-----------------------------
 0  | 18   | 1.0    | [0.0, 10.0, 0.5] | 1.0     | [18.0, 1.0, 0.0, 10.0, 0.5]
```

图 12-8 用户特征 VectorAssembler 示例

VectorAssembler 需要通过如下代码导入模块。

```
from pyspark.ml.feature import VectorAssembler
```

VectorAssembler 参数说明：

- inputCols：要进行转换的字段名（列名），可以传多列
- outputCol：转换后输出的字段名（列名）

VectorAssembler 使用方法如图 12-9 代码所示。

```
In [26]: dataset = spark.createDataFrame(
             [(0, 18, 1.0, Vectors.dense([0.0, 10.0, 0.5]), 1.0)],
             ["id", "hour", "mobile", "userFeatures", "clicked"])

         assembler = VectorAssembler(
             inputCols=["hour", "mobile", "userFeatures"],
             outputCol="features")

         output = assembler.transform(dataset)
         output.show(truncate=False)
```

1. 创建DataFrame数据集

2. 创建VectorAssembler

3. 使用transform方法转换

```
+---+----+------+--------------+-------+-----------------------+
|id |hour|mobile|userFeatures  |clicked|features               |
+---+----+------+--------------+-------+-----------------------+
|0  |18  |1.0   |[0.0,10.0,0.5]|1.0    |[18.0,1.0,0.0,10.0,0.5]|
+---+----+------+--------------+-------+-----------------------+
```

图 12-9

可以通过 take()方法或者 printSchema()方法打印输出数据集的数据结构，查看 feature 字段类型，如图 12-10 所示。

```
In [28]: output.take(1)
Out[28]: [Row(id=0, hour=18, mobile=1.0, userFeatures=DenseVector([0.0, 10.0, 0.5]), clicked=1.0, features=DenseVector
         ([18.0, 1.0, 0.0, 10.0, 0.5]))]

In [27]: output.printSchema()

root
 |-- id: long (nullable = true)
 |-- hour: long (nullable = true)
 |-- mobile: double (nullable = true)
 |-- userFeatures: vector (nullable = true)
 |-- clicked: double (nullable = true)
 |-- features: vector (nullable = true)
```

图 12-10 打印 VectorAssembler 结果

12.2.4 MinMaxScaler

MinMaxScaler 用于将数据缩放至 0~1 之间。MinMaxScaler 转换 Vector 行数据集，将每个特征重新缩放到特定范围（通常为 [0, 1]）。

MinMaxScaler 需要通过如下代码导入模块。

```
from pyspark.ml.feature import MinMaxScaler
```

MinMaxScaler 参数说明：

- min：默认为 0.0，转换后的下界，由所有特征共享
- max：默认为 1.0，转换后的上限，由所有特征共享
- inputCol：要进行转换的字段名（列名）
- outputCol：转换后输出的字段名（列名）

MinMaxScaler 使用方法如图 12-11 代码所示。

```
In [33]: dataFrame = spark.createDataFrame([
             (0, Vectors.dense([1.0, 0.1, -1.0]),),
             (1, Vectors.dense([2.0, 1.1, 1.0]),),
             (2, Vectors.dense([3.0, 10.1, 3.0]),)
         ], ["id", "features"])

         scaler = MinMaxScaler(inputCol="features",
                               outputCol="scaledFeatures")

         scalerModel = scaler.fit(dataFrame)
         scaledData = scalerModel.transform(dataFrame)

         scaledData.select("features", "scaledFeatures").show()

+--------------+--------------+
|      features|scaledFeatures|
+--------------+--------------+
|[1.0,0.1,-1.0]|[0.0,0.0,0.0]|
| [2.0,1.1,1.0]|[0.5,0.1,0.5]|
|[3.0,10.1,3.0]|[1.0,1.0,1.0]|
+--------------+--------------+
```

1. 创建DataFrame数据集
2. 创建MinMaxScaler
3.使用 fit()方法生成Transformer
4. 使用transform()方法进行转换

图 12-11 MinMaxScaler 使用方法

同样，我们可以通过 take()方法打印输出数据集的数据结构，查看 scaledFeatures 字段类型，如图 12-12 代码所示。

```
In [34]: scaledData.take(1)
Out[34]: [Row(id=0, features=DenseVector([1.0, 0.1, -1.0]), scaledFeatures=DenseVector([0.0, 0.0, 0.0]))]
```

图 12-12 打印 MinMaxScaler 输出结果

12.3 Pipeline 逻辑回归实战

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

12.3.1 创建数据集

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 12-13 所示。

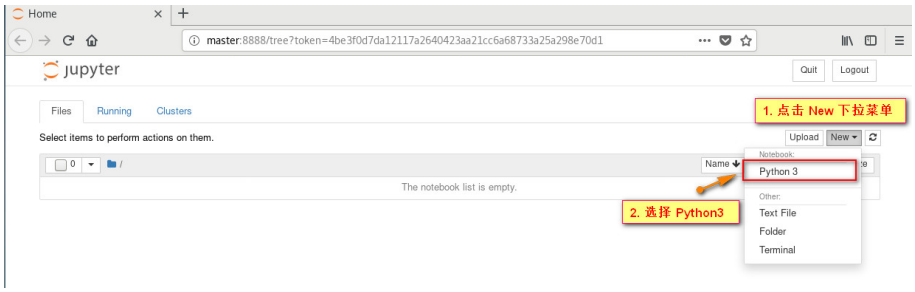

图 12-13 新建 Notebook 文件

步骤 2：重命名 Notebook

单击【Untitled】，输入“Chapter12”，并确认，为 Notebook 重命名，如图 12-14 所示。

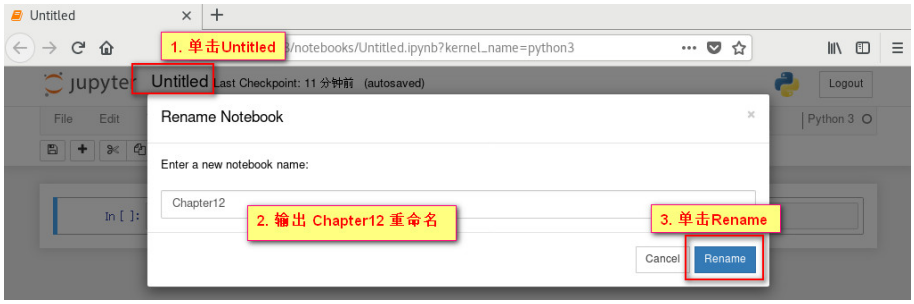

图 12-14 重命名 Notebook 文件

步骤 3：获取 Notebook 路径

在 Notebook 代码行中输入如下代码，导入本节需要使用到的库，并且获取 Notebook 所在路径并打印，如图 12-15 所示。

```
In [1]: # coding: utf-8
        from pyspark.sql.types import *
        import pyspark.sql.functions as F
```

导入相关依赖库

```
In [2]: pwd = !pwd
        data_path = "file://" + list(pwd)[0] + '/data/'
        print(data_path)

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

图 12-15 初始化代码

步骤 4：创建 SparkSession Application

创建 SparkSession 代码如图 12-16 所示。

```
In [49]: from pyspark.sql.types import *
         from pyspark.sql import SparkSession

In [54]: spark = SparkSession\
             .builder\
             .appName('Pipeline-ML')\
             .getOrCreate()

In [55]: spark

Out[55]: SparkSession - hive
         SparkContext

         Spark UI
         Version
         v2.4.3
         Master
         local
         AppName
         Pipeline-ML
```

图 12-16 读取 PM2.5 数据集

步骤 5：读取数据集

使用 spark.read.csv 函数读取 bank.csv 数据文件，如图 12-17 所示。

```
In [112]: df = spark.read.csv(data_path+"bank.csv",
                              header=True,
                              inferSchema=True)
```

图 12-17 读取 PM2.5 数据集

步骤 6：查看数据结构

使用 printSchema()函数查看数据集的字段结构，如图 12-18 所示。

```
In [113]: df.printSchema()

root
 |-- age: integer (nullable = true)
 |-- job: string (nullable = true)
 |-- marital: string (nullable = true)
 |-- education: string (nullable = true)
 |-- default: string (nullable = true)
 |-- balance: integer (nullable = true)
 |-- housing: string (nullable = true)
 |-- loan: string (nullable = true)
 |-- contact: string (nullable = true)
 |-- day: integer (nullable = true)
 |-- month: string (nullable = true)
 |-- duration: integer (nullable = true)
 |-- campaign: integer (nullable = true)
 |-- pdays: integer (nullable = true)
 |-- previous: integer (nullable = true)
 |-- poutcome: string (nullable = true)
 |-- deposit: string (nullable = true)
```

图 12-18 查看数据集字段结构

步骤 7：数据准备

我们需要将数据集中不需要的字段进行删除操作，然后再对数据集进行切分训练集和测试集，代码如图 12-19 所示。

```
In [127]: df = df.drop('day','month')
          train_df, test_df = df.randomSplit([0.7, 0.3], seed = 666)

In [128]: print("Train Dataset Num :", train_df.count())
          print("Test Dataset Num :", test_df.count())

          Train Dataset Num : 7821
          Test Dataset Num : 3341
```

1.删除不需要的字段

2.按7:3切分训练集和测试集

3. 打印查看数据集数量

图 12-19 查看数据集字段结构

12.3.2 建立 ML Pipeline

本章我们建立 ML Pipeline 流程包含 4 个阶段、

1. StringIndexer：将字符串字段进行转换。
2. OneHotEncoderEstimator：将数字编码字段进行 OneHot 转换。
3. VectorAssembler：将所有的特征字段整合成 Vector。
4. LogisticRegression：逻辑回归进行预测。

步骤 1：导入相关模块

导入模块具体代码如图 12-20 所示。

```
In [129]: from pyspark.ml import Pipeline
          from pyspark.ml.feature import StringIndexer, VectorAssembler,OneHotEncoderEstimator
          from pyspark.ml.classification import LogisticRegression
```

图 12-20 模块具体代码

步骤 2：建立类别/数值特征

由于数据集中包含大量的离散类别型特征和连续数值型特征，这里我们需要进行区分，便于后续对它们进行统一处理，如图 12-21 所示。

```
In [102]: categoricalColumns = ['job', 'marital', 'education', 'default', 'housing', 'loan', 'contact', 'poutcome']
          numericCols = ['age', 'balance', 'duration', 'campaign', 'pdays', 'previous']
```

图 12-21 创建类别特征列和数值特征列

步骤 3：创建 pipeline

这里我们需要对类别特征进行 StringIndexer 和 OneHot 编码，所以我们遍历所有类别特征进行处理，建立 pipeline 代码如下图 12-22 所示。

```
In [103]: stages = []   # 1. 创建stages

for col in categoricalColumns:   # 2. 遍历类别，特征进行StringIndexer和OneHot编码
    stringIndexer = StringIndexer(inputCol = col, outputCol = col + 'Index')
    encoder = OneHotEncoderEstimator(inputCols=[stringIndexer.getOutputCol()], outputCols=[col + "classVec"])
    stages += [stringIndexer, encoder]

label_stringIdx = StringIndexer(inputCol = 'deposit', outputCol = 'label')
stages += [label_stringIdx]

assemblerInputs = [c + "classVec" for c in categoricalColumns] + numericCols   # 3. 将所有特征整合成Vector
assembler = VectorAssembler(inputCols=assemblerInputs, outputCol="features")
stages += [assembler]

lr = LogisticRegression(featuresCol = 'features', labelCol = 'label', maxIter=10)   # 4. 创建LR模型，添加stages
stages += [lr]
```

图 12-22 建立 StringIndexer、OneHot 和 VectorAssembler pipeline

步骤 4：建立 pipeline

使用 Pipeline 函数建立 pipeline，代码如下图 12-23 所示。

```
In [133]: pipeline = Pipeline(stages = stages)
```

图 12-23 创建 pipeline 并添加 stages

步骤 5：查看 pipeline 阶段

在步骤 4 完成 pipeline 的创建工作之后，使用 getStages()函数可以查看 pipeline 包含哪些 stage，如下图 12-24 所示。

```
In [135]: pipeline.getStages()

Out[135]: [StringIndexer_8b3b68275586,
           OneHotEncoderEstimator_c64a03f5f9d3,
           StringIndexer_a0f4bc18c094,
           OneHotEncoderEstimator_f43d92145819,
           StringIndexer_aa394a87cc39,
           OneHotEncoderEstimator_688ee7027d18,
           StringIndexer_f1626192a44f,
           OneHotEncoderEstimator_19b52f34d58a,
           StringIndexer_3d6190b7cff4,
           OneHotEncoderEstimator_e2f68d5e2e42,
           StringIndexer_aa6f2bb4b051,
           OneHotEncoderEstimator_d1345f720427,
           StringIndexer_587b696899a9,
           OneHotEncoderEstimator_4506790eb53a,
           StringIndexer_6db25f74f00e,
           OneHotEncoderEstimator_4ab28a628c95,
           StringIndexer_96c5a7321a68,
           VectorAssembler_5a8be98fd15d,
           LogisticRegression_db2ac13a3ab6]
```

图 12-24 查看 pipeline 中各个 stage

12.3.3 模型训练

在上一小节中，我们完成了 pipeline 的创建，并通过 getStages()函数查看了 pipeline 所有的 stage 情况。接下来，我们将使用上一小节创建的 pipeline.fit()对 train_df 数据集进行训练工作。

步骤 1：fit()训练模型

进行训练之后，将会输出 pipelineModel，它包含 pipeline 的所有阶段，如下图 12-25 所示。

```
In [136]: pipelineModel = pipeline.fit(train_df)
```

图 12-25 使用 fit 训练模型

步骤 2：查看训练后的 LR 模型

LR 逻辑回归是 pipeline 的最后一个 stage，所以使用如下代码可以查看训练完成后的 LR 模型，如图 12-26 所示。

```
In [137]: pipelineModel.stages[-1]
Out[137]: LogisticRegressionModel: uid = LogisticRegression_db2ac13a3ab6, numClasses = 2, numFeatures = 30
```

图 12-26 查看 LR 模型 stage 内容

12.3.4 使用模型预测

在上一小节中，我们完成了 pipeline 的建立与训练工作，接下来我们将用前面训练完成的 pipelineModel 进行 test_df 的预测。

步骤 1：使用 transform()进行预测

pipelineModel 使用 transform()可以对训练集进行预测，如下图 12-27 所示。

```
In [117]: df_out = pipelineModel.transform(test_df)
```

图 12-27 使用 transform 进行预测

步骤 2：查看预测结果

```
In [122]: df_out.select('features', 'label', 'rawPrediction', 'prediction', 'probability').show(10)
```

```
+--------------------+-----+--------------------+----------+--------------------+
|            features|label|       rawPrediction|prediction|         probability|
+--------------------+-----+--------------------+----------+--------------------+
|(30,[7,12,16,17,1...|  1.0|[-0.6298499099748...|       1.0|[0.34754457104364...|
|(30,[7,12,13,16,1...|  1.0|[-1.0056204029849...|       1.0|[0.26783781916319...|
|(30,[7,12,16,17,1...|  0.0|[-1.0679101456932...|       1.0|[0.25580071979005...|
|(30,[7,12,16,17,1...|  1.0|[-3.8353272081235...|       1.0|[0.02113781576069...|
|(30,[7,12,13,16,1...|  1.0|[-0.6550647423974...|       1.0|[0.34184911852098...|
|(30,[1,12,13,16,1...|  0.0|[-1.7658964151801...|       1.0|[0.14605339240666...|
|(30,[7,12,15,16,1...|  1.0|[-2.9785282464856...|       1.0|[0.04840537602580...|
|(30,[7,12,13,16,1...|  1.0|[-3.7014900436566...|       1.0|[0.02409196334164...|
|(30,[7,12,13,16,1...|  1.0|[-0.5852110096154...|       1.0|[0.35773442391132...|
|(30,[7,12,13,16,1...|  1.0|[-3.1212733901782...|       1.0|[0.04223822787354...|
+--------------------+-----+--------------------+----------+--------------------+
only showing top 10 rows
```

图 12-28 查看预测结果

上图中 prediction 列即为 LR 模型对测试集的预测，label 为数据的真实标签。

12.3.5 模型评估

在前面小节中，我们完成了 pipeline 的建立、pipeline 的训练和预测工作，接下来，我们希望通过理量化指标对模型预测的效果进行评估。

步骤 1：导入模块

```
from pyspark.ml.evaluation import BinaryClassificationEvaluator
```

由于上述代码使用 LR 完成的二分类任务，所以导入 BinaryClassificationEvaluator。

步骤 2：创建 BinaryClassification Evaluator

```
In [124]: evaluator = BinaryClassificationEvaluator()
```

批注 [雷2]: 添加图片说明

图 12-29 创建 BinaryClassification Evaluator

步骤 3：计算 AUC

LR 模型通过计算 AUC 来评估模型的效果，如下图 12-27 所示。

```
In [125]: print('AUC: Logistic Regression   : ', evaluator.evaluate(df_out))
          AUC: Logistic Regression   :  0.8835721328251214
```

图 12-30 评估模型的效果

可以看到，模型评估 AUC 值为 0.88 左右，说明模型效果较为理想。

★回顾思考★

01 Spark ML Pipeline 包含哪些组件？

答：

（1） DataFrame

（2） Transformer

（3） Estimator

（4） Pipeline

02 Spark ML Pipepline 使用的 StringIndexer、OneHotEncoder 在哪个库中？

答：

```
org.apache.spark.ml.feature.*
```

批注 [雷3]: 语意不明，需审核或者修改

★练习★

一、选择题

1. 不属于 Spark ML Pipeline 组件的是（ ）

A. DataFrame

B. Transformer
C. Estimator
D. RDD

2. 完成将所有字段整合成一个字段的模块是（ ）
A. OneHotEncoder
B. StringIndexer
C. VectorAssembler
D. MinMaxScaler

3. 建立 ML Pipeline 使用的函数是（ ）
A. Pipeline ()
B. MakePipeline ()
C. pipeline ()
D. Pipline ()

二、判断题

1. ML Pipeline 的组件包含 DataSet 和 RDD。（×）
2. OneHotEncoder 用于将一个数值类型的特征字段转换为多个字段的 Vector。（√）

三、实战练习

任务 1：手动实现 12.2 节中各个方法的代码。
请参考 12.2 小节中的代码。
任务 2：完成基于建立 ML Pipeline 的逻辑回归代码实战。
请参考本书配套第 12 章的 Notebook 代码文件

批注 [雷4]: 需完成答案

批注 [HZ5R4]: 这个是在代码文件里，是 .ipynb 的代码文件，没有办法放在 word 里。

本章小结

本章详细介绍了 Spark ML Pipeline 的理论知识，并介绍了在建立 ML Pipeline 中常用的函数方法，如 StringIndexer、OneHotEncoder 等，并均通过代码的方式进行了展示。最后本章通过逻辑回归代码实战的方式，完整的进行了 Spark ML Pipeline 的流程，包括数据预处理、数据集切分、Pipeline 建立、模型训练、模型预测与评估。