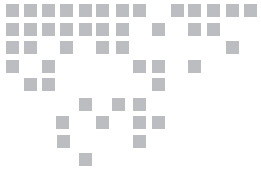

Chapter 3 第 3 章

ML Pipeline 原理与实战

Spark MLlib 是 Spark 的重要组成部分，也是最早推出的库之一，其基于 RDD 的 API，算法比较丰富，比较稳定，也比较好用。但是如果目标数据集结构复杂需要多次处理，或者是对新数据需要结合多个已经训练好的单个模型进行综合计算时，使用 MLlib 将会让程序结构复杂，甚至难于理解和实现。为改变这一局限性，从 Spark 1.2 版本之后引入了 ML Pipeline，经过多个版本的发展，Spark ML 克服了 MLlib 在处理复杂机器学习问题的一些不足（如工作比较复杂，流程不清晰等），向用户提供基于 DataFrame 之上的更加高层次的 API 库，以更加方便的构建复杂的机器学习工作流式应用，使整个机器学习过程变得更加易用、简洁、规范和高效。Spark 的 Pipeline 与 Scikit 中 Pipeline 的功能相近、理念相同。本章主要介绍 Spark ML 中 Pipeline 的有关内容。

本章主要介绍 ML Pipeline 的相关内容，包括：

- ❏ Pipeline 简介
- ❏ DataFrame
- ❏ 构成 Pipeline 的一些组件
- ❏ 介绍 pipeline 的一般原理
- ❏ 使用 pipeline 的几个实例

3.1 Pipeline 简介

ML 提倡使用 Pipeline，一般翻译为流水线，以便将多种算法更容易地组合成单个流水线或工作流程。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、一个特征提取工具，都可以抽象为一个 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。与 Pieline 相关的概念有：DataFrame、Transformer、Estimator、Parameter 等。

3.2　DataFrame

说到 DataFrame，使用过 Pandas、R 的朋友应该比较熟悉，它是一种类似于关系型数据库的表，使用和维护都非常方便。Spark 的 DataFrame 是基于早期版本中的 SchemaRDD，所以使用的是分布式大数据处理的场景。Spark DataFrame 以 RDD 为基础，但是带有 Schema 信息，可以从常规隐式地或明确地创建 RDD，它类似于传统数据库中的二维表格。它被 ML Pipeline 用来存储源数据。DataFrame 可以保存各种类型的数据，如我们可以把特征向量存储在 DataFrame 的一列中，这样用起来非常方便。

以下通过一个实例来说明 DataFrame 的创建、操作等内容：

```
// 以一个简单客户文件 customer.txt 为例，含 name,age,gender 三列
$cat customer.txt
name,age,gender
吴凡,25,F
张海燕,28,F
张宏,30,M
刘婷,29,F
高峰,31,M
// 创建 DataFrame
import org.apache.spark.sql.SparkSession
import org.apache.spark.sql.DataFrame

val spark = SparkSession.builder()
.appName("Spark SQL basic example")
.config("spark.some.config.option", "some-value")
.getOrCreate()
// 创建 DataFrame
val df1 = spark.read.option("header", true).format("csv").load("file:///home/
          hadoop/data/customer.csv")

// 转换字符类型
val df2 = df1.select(
    df1("name").cast("String"),
    df1("age").cast("Double"),
    df1("gender").cast("String"))
// 显示 df2 的 Schema
```

```
df2.printSchema()
root
  |-- name: string (nullable = true)
  |-- age: double (nullable = true)
  |-- gender: string (nullable = true)
// 创建视图，后续可以把视图作为表样使用
df2.createOrReplaceTempView("customer")

// 查询
val cust1 = spark.sql("SELECT * FROM customer WHERE age BETWEEN 30 AND 35")
cust1.limit(5).show
+----+----+------+
|name| age|gender|
+----+----+------+
|  张宏|30.0|     M|
|  高峰|31.0|     M|
| 赵建军|32.0|     M|
|  李俊|32.0|     M|
| 孙达荣|33.0|     M|
+----+----+------+
val cust2 = spark.sql("SELECT * FROM customer WHERE gender like 'M'")
cust2.limit(5).show
+----+----+------+
|name| age|gender|
+----+----+------+
|  张宏|30.0|     M|
|  高峰|31.0|     M|
|  王华|40.0|     M|
| 赵建军|32.0|     M|
|  李俊|32.0|     M|
+----+----+------+
```

3.3 Pipeline 组件

前面提到 Pipeline 组件主要包括 Transformer 和 Estimator，下面来详细介绍。

1. 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。

2. 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。从技术上来说，估计器通过调用 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，逻辑回归就是一个估计器，通过调用 fit() 来产生一个逻辑回归模型。

3.4　Pipeline 原理

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。

流水线由一系列有顺序的阶段指定，每个状态的运行是有顺序的，输入的 DataFrame 通过每个阶段进行改变。在转换器阶段，transform() 方法被调用于 DataFrame 上。对于估计器阶段，fit() 方法被调用来产生一个转换器，然后该转换器的 transform() 方法被调用在 DataFrame 上。图 3-1 简单说明了文档处理工作流的运行过程。

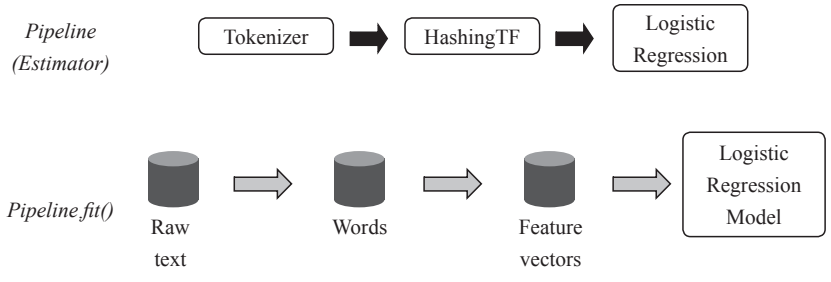

图 3-1　Pipeline 在训练数据上的流程

在图 3-1 中，第一行代表流水线处理的三个阶段。第一、二个阶段是转换器，第三个逻辑回归是估计器。底下一行代表流水线中的数据流，圆筒指 DataFrame。流水线的 fit() 方法被调用于原始的 DataFrame 中，里面包含原始的文档和标签。分词器的 transform() 方法将原始文档分为词语，添加新的词语列到 DataFrame 中。哈希处理的 transform() 方法将词语列转换为特征向量，添加新的向量列到 DataFrame 中。然后，因为逻辑回归是估计器，流水线先调用逻辑回归的 fit() 方法来产生逻辑回归模型。如果流水线还有其他更多阶段，在将 DataFrame 传入下一个阶段之前，流水线会先调用逻辑回归模型的 transform() 方法。

整个流水线是一个估计器。所以当流水线的 fit() 方法运行后，会产生一个流水线模型，流水线模型是转换器。流水线模型会在测试时被调用，图 3-2 说明了它的用法。

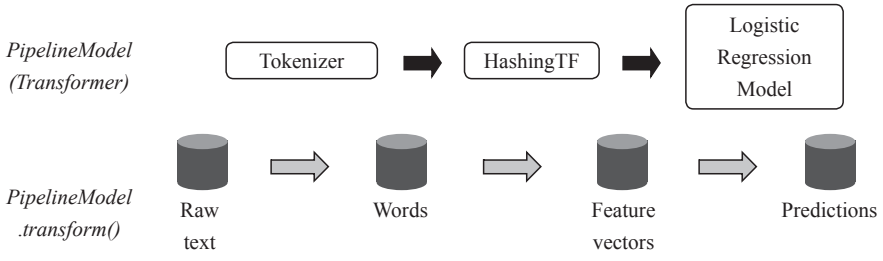

图 3-2 Pipeline 在测试数据上的流程

在图 3-2 中，流水线模型和原始流水线有同样数目的阶段，然而原始流水线中的估计器此时变为了转换器。当流水线模型的 transform() 方法被调用于测试数据集时，数据依次经过流水线的各个阶段。每个阶段的 transform() 方法更新数据集，并将之传到下个阶段。

流水线和流水线模型有助于确认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，可用图 3-3 来表达。

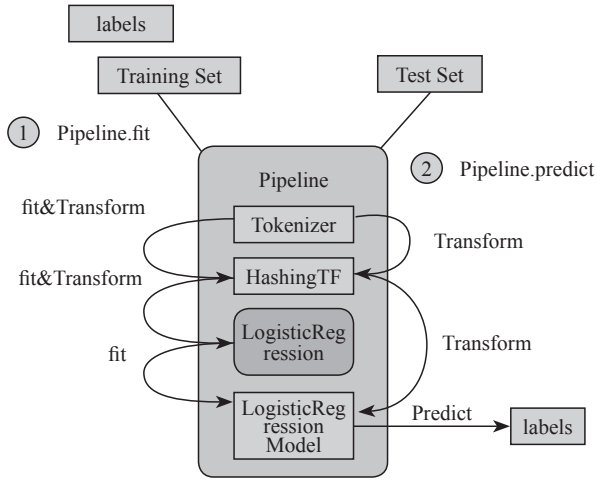

图 3-3 Spark pipeline 流程图

其中 Pipeline 及 LogisticRegression 为 Estimator，Tokenizer、HashingTF、LogisticRegression Model 为 Transformer。

3.5 Pipeline 实例

3.5.1 使用 Estimator、Transformer 和 Param 的实例

在机器学习整个过程中，特征转换、特征选择、派生特征等工作，一般需要占据大部分时间，现在 ML 提供了很多 Transformer，如 OneHotEncoder、StringIndexer、PCA、

Bucketizer、Word2vec 等，利用这些函数可极大提高工作效率。

以下通过实例说明如何使用 ML 库中的 Estimaor、Transformer 和 Param 等。

```
import org.apache.spark.ml.classification.LogisticRegression
import org.apache.spark.ml.linalg.{Vector, Vectors}
import org.apache.spark.ml.param.ParamMap
import org.apache.spark.sql.Row

// 从（标识、特征）元组开始训练数据
val training = spark.createDataFrame(Seq(
    (1.0, Vectors.dense(0.0, 1.1, 0.1)),
    (0.0, Vectors.dense(2.0, 1.0, -1.0)),
    (0.0, Vectors.dense(2.0, 1.3, 1.0)),
    (1.0, Vectors.dense(0.0, 1.2, -0.5))
)).toDF("label", "features")

// 创建一个 LogisticRegression 实例，该实例是一个估计器
val lr = new LogisticRegression()
// 打印参数、文档和任何默认值
println("LogisticRegression parameters:\n" + lr.explainParams() + "\n")

// 我们可以使用 setter 方法设置参数
lr.setMaxIter(10)
    .setRegParam(0.01)

// 训练 LogisticRegression 模型，这里使用了存储在 lr 的参数
val model1 = lr.fit(training)
// 因为 model 是模型（即由估计器生成的转换器），我们可以查看它在 fit() 中使用的参数。
// 打印参数（名称：值）对，其中 LogisticRegression 实例的名称是唯一的 ID
//
println("Model 1 was fit using parameters: " + model1.parent.extractParamMap)

// 我们可以用 ParamMap 指令参数，它支持几种指定参数的方法
val paramMap = ParamMap(lr.maxIter -> 20)
    .put(lr.maxIter, 30)  // 指定 1 个参数，这会覆盖原来的 maxIter
    .put(lr.regParam -> 0.1, lr.threshold -> 0.55)  // 指定多个参数

// 也可以组合 paramMap
val paramMap2 = ParamMap(lr.probabilityCol -> "myProbability")  // 修改输出列名
val paramMapCombined = paramMap ++ paramMap2

// 现在使用 paramMapCombined 参数学习一个新的模型
// paramMapCombined 覆盖之前通过 lr.set* 方法设置的所有参数
val model2 = lr.fit(training, paramMapCombined)
println("Model 2 was fit using parameters: " + model2.parent.extractParamMap)

// 准备测试数据
val test = spark.createDataFrame(Seq(
    (1.0, Vectors.dense(-1.0, 1.5, 1.3)),
    (0.0, Vectors.dense(3.0, 2.0, -0.1)),
```

```
    (1.0, Vectors.dense(0.0, 2.2, -1.5))
)).toDF("label", "features")

// 使用 Transformer.transform() 方法对测试数据进行预测。
// LogisticRegression.transform 将仅使用“特征”列。
// 请注意 ,model2.transform() 输出一个 "myProbability" 列 , 而不是通常我们先前通过
// lr.probabilityCol 参数重命名的 'Probability 列。
model2.transform(test)
    .select("features", "label", "myProbability", "prediction")
    .collect()
    .foreach { case Row(features: Vector, label: Double, prob: Vector,
prediction: Double) =>
    println(s"($features, $label) -> prob=$prob, prediction=$prediction")
    }
```

3.5.2 ML 使用 Pipeline 的实例

要构建一个 Pipeline，首先我们需要定义 Pipeline 中的各个 Pipeline Stage，如分词、计算 TF-IDF 及训练逻辑回归模型等。这些 Transformer 和 Estimator 创建后，我们就可以按照处理流程组成 PipelineStages，并创建一个 Pipeline，如 val pipeline = new Pipeline().setStages(Array(stage1,stage2,…))。然后，把训练数据集作为参数并调用 Pipeline 实例的 fit() 方法，之后，将以流程的方式来处理原训练数据。以下是一个具体使用 Pipeline 的实例。

```
import org.apache.spark.ml.{Pipeline, PipelineModel}
import org.apache.spark.ml.classification.LogisticRegression
import org.apache.spark.ml.feature.{HashingTF, Tokenizer}
import org.apache.spark.ml.linalg.Vector
import org.apache.spark.sql.Row

// 准备训练文档 (id, 内容 , 标签 )
val training = spark.createDataFrame(Seq(
    (0L, "a b c d e spark", 1.0),
    (1L, "b d", 0.0),
    (2L, "spark f g h", 1.0),
    (3L, "hadoop mapreduce", 0.0)
)).toDF("id", "text", "label")

// 配置 ML Pipeline, 由三个 stage 组成 ,tokenizer、hashingTF 和 lr
val tokenizer = new Tokenizer()
    .setInputCol("text")
    .setOutputCol("words")
val hashingTF = new HashingTF()
    .setNumFeatures(1000)
    .setInputCol(tokenizer.getOutputCol)
    .setOutputCol("features")
val lr = new LogisticRegression()
  .setMaxIter(10)
  .setRegParam(0.001)
```

```
    val pipeline = new Pipeline()
        .setStages(Array(tokenizer, hashingTF, lr))

    //在训练数据集上使用 Pipeline
    val model = pipeline.fit(training)

    // Now we can optionally save the fitted pipeline to disk
    //现在可以保存安装好的流水线到磁盘上
    model.write.overwrite().save("/tmp/spark-logistic-regression-model")

    //现在可以保存未安装好的 Pipeline 保存到磁盘上
    pipeline.write.overwrite().save("/tmp/unfit-lr-model")

    // 装载模型
    val sameModel = PipelineModel.load("/tmp/spark-logistic-regression-model")

    //准备测试文档，不包含标签 (id, text) .
    val test = spark.createDataFrame(Seq(
        (4L, "spark i j k"),
        (5L, "l m n"),
        (6L, "spark hadoop spark"),
        (7L, "apache hadoop")
    )).toDF("id", "text")
    //在测试文档上做出预测
    model.transform(test)
        .select("id", "text", "probability", "prediction")
        .collect()
        .foreach { case Row(id: Long, text: String, prob: Vector, prediction: Double)
=>
            println(s"($id, $text) --> prob=$prob, prediction=$prediction")
        }
```

3.6　小结

本章主要介绍了流水线（pipeline）的基本概念，以流水线的两个组件（Transformer 和 Estimator），它们是构成 Pipeline 的 Stage，把这些 Stage 按照一定次序组装到 Pipeline 上，就构成一个流水线，这些 Stage 包括特征转换、特征选择、模型训练等任务，然后通过几个实例具体说明 Pipeline 的创建及使用。流水线是一项重要内容，下一章将主要介绍构成 Pipeline 的一些 Stage。熟练使用这些 Stage 有助于提升我们的开发效率。